Advanced Tricks, Tips, and Techniques for Java Programming

Java

案例精粹150例

张怀庆 编著



冶金工业出版社

Downloads By http://down.liehuo.net

前　言

一、关于 Java

Java(tm)是由美国升阳电脑公司（Sun Microsystems，Inc）所发展出的第一种能在国际互联网络（Internet）上具有「硬件/软件中立性」交互能力的程序语言。Java 一推出即引起世界各种媒体的广泛注意与讨论，如 Time、Business Weekly、CNN 等，可见其功能及魅力，事实上，以 Java 的功能，势必在信息界掀起另一番高潮，对于网络的发展，又将迈入另一个新的纪元。Java 是一种软件技术，在 Sun 的 Java 语言白皮书中明白地说明 Java 语言的特点：简单的（Simple）、面向对象的（Object-Oriented）、分布式的（Distributed）、健壮的（Robust）、安全的（Secure）、结构中立的（Architecture Neutral）、可移植的（Portable）、解释的（Interpreted）、高效能的（High Performance）、多线程的（Multi threaded）、动态的（Dynamic）。

1998 年，Sun 发布了 Java 2 平台。在 Java 的发展史上，Java 2 是一个里程碑式的产品。Java 2 不但在性能上有很大提高，还提供了一个安全、灵活的新模式，以及一套完整的应用软件开发接口，同年，Sun 在 Javaone 大会上发布了 EJB1.0 规范，为建立分布式商务对象系统提供了坚实的结构体系基础。从此 Java 开始走向企业级应用平台。Sun 确定的战略目标是将 Java 发展成为三个系统：针对嵌入式设备的 J2ME、针对个人 PC 的 J2SE、企业级服务器端平台的 J2EE。

2000 年 Sun 发布了 J2ME。J2ME 是针对嵌入式的 Java 平台，随后 Sun 又发布了专门针对移动通讯工具的 MIDP 开发规范以作为 J2ME 的补充。到 2004 年，J2ME 技术、电子邮件、HTML 衍生产品已经成为手持移动设备的主要开发平台。

2001 年 1 月 Java 的企业版 J2EE 发布，J2EE 是分布式企业软件组件架构的规范，具有 Web 性能、灵活性、简化的集成性、便捷性以及 J2EE 服务器之间的互操作性。

2001 年 9 月 J2SDK1.4Beta2 发布。Sun 制定 Java 的发展方向后，JDK1.2 以后的版本均称为 J2SDK，这些版本都支持 Java 2 平台标准，同时为了支持业界的开放源代码运动，Sun 公布了部分版本的源代码，可以在网上直接下载。Sun 还发布了 J2SDK1.5 版本，在这个版本中 Sun 在语言和技术都作了重大改进，Java 的功能变得更为强大。

二、本书结构

本书内容结构安排如下：

第 1 章：Java 编程初步。介绍了 Java 面向对象编程的基础知识及 Java 数据结构的编程。

第 2 章：Java 多线程编程。介绍了利用 Java 的多线程功能编程，并介绍了处理多个任务的应用程序，还介绍了几个多线程的著名问题的实现程序。

第 3 章：Java 输入/输出编程。介绍了 Java 的输入、输出类的应用，包括文件的读写、压缩以及流、管道等的操作，目录示例器、对象次第读写等编程知识。

第 4 章：Java Applet 编程。介绍了 Applet 小程序的开发，包括简单的 Applet 程序、文

本域、多行文本域和它们的触发事件等。

第 5 章：Java Swing 图形界面编程。介绍了利用 Swing 组件进行界面设计。

第 6 章：Java 多媒体和图形图像处理编程。介绍了 Java 在多媒体方面对音频和图像的处理。

第 7 章：Java 网络编程。介绍了 Java 获取网络资源、套接字、服务器/客户机。

第 8 章：Java 数据库编程。介绍了利用 JDBC 进行数据库连接和数据库编程的 7 个实例。

第 9 章：Java B/S 结构编程。通过 14 个实例介绍了 JSP、JavaBean、Servlet 等技术的编程应用。

第 10 章：Java 的 Web Service 编程。通过 6 个实例介绍了 XML 和 SOAP 编程。

第 11 章：EJB 编程。通过 7 个实例介绍了各种 EJB 的编程，如无状态会话 Bean、有状态会话 Bean、银行存取 BMP 等。

第 12 章：Java 安全性编程。介绍了各种加、解密和各种安全机制编程技术。

第 13 章：J2ME 编程。介绍了 J2ME 的框架和简单应用，包括 Command 对象应用、查看内存利用情况、菜单设计、记录存储、使用定时器、无线 HTTP 连接等。

第 14 章：Java 其他编程。介绍了 JNI 和一些与系统有关的编程技术的应用。

第 15 章：Java 工具和服务器配置与集成。介绍了 Java 的各种开发环境的安装和配置。

三、本书特点

本书内容丰富、结构合理，叙述简洁明了，有很强的实用性，以实例的形式向读者展示了 Java 2 编程技术的精髓，既注重面向对象软件设计的基础，又体现了 Java 2 的新技术。

四、本书适用对象

本书例题典型、结构合理、思路清晰，既可作为大专院校相关专业及 Java 培训班的教材，也可作为具有一定 Java 编程知识的自学者的参考用书，同时还可以作为从事软件设计的工程人员的参考用书。

本书针对 Java 2 的各种技术进行了全面的介绍，要求读者初步掌握 Java 程序设计的基础知识，可将此书作为 Java 入门与提高的参考书，可以帮助读者迅速掌握 Java 的各种技术的应用方法。本书的实例都通过编译并给出了运行结果，如有问题，敬请读者批评指正。

由于编者水平有限，编写时间仓促，缺点和错误在所难免，希望广大读者批评指正。

虽然经过严格的审核、精细的编辑，本书在质量上有了一定的保障，但我们的目标是力求尽善尽美，欢迎广大读者和专家对我们的工作提出宝贵建议，联系方法如下：

电子邮件：service@cnbook.net

网址：www.cnbook.net

本书涉及的源代码可在该网站的下载中心免费下载。此外，该网站还有一些其他相关书籍的介绍，可以方便读者选购参考。

编　者

2004 年 12 月

目　录

第 1 章　Java 编程初步 1

实例 1　Hello world! 1
实例 2　Integer 类 2
实例 3　显示最大值 3
实例 4　计算某年某月的天数 4
实例 5　数值运算 5
实例 6　日历表 6
实例 7　Character 类 7
实例 8　String 类 8
实例 9　StirngBuffer 类 9
实例 10　数组 10
实例 11　类 11
实例 12　Class 类 12
实例 13　观察垃圾回收过程 14
实例 14　继承 17
实例 15　动态绑定 17
实例 16　接口 18
实例 17　异常处理 19
实例 18　冒泡排序 20
实例 19　Hanio 塔问题 21
实例 20　链表 21
实例 21　用 LinkedList 实现栈 24
实例 22　Vector 类 26
实例 23　Properties 类 27
实例 24　ArrayList 类 28
实例 25　线程初探 29
小结 30

第 2 章　Java 多线程编程 32

实例 26　简单的多线程应用 32
实例 27　简单动画 34
实例 28　模拟接力 35
实例 29　线程树 37

实例 30 生产者 - 消费者问题 39
实例 31 死锁 42
实例 32 哲学家就餐问题 43
小结 46

第 3 章 Java 输入/输出编程 47

实例 33 文件的读取 47
实例 34 写入文件 48
实例 35 标准 I/O 重定向 49
实例 36 流操作 49
实例 37 管道操作 54
实例 38 目录列示器 55
实例 39 目录的检查与修改 56
实例 40 压缩文件 58
实例 41 对象次第读写 59
小结 61

第 4 章 Java Applet 编程 62

实例 42 简单的 Applet 程序 62
实例 43 文本域 64
实例 44 文本域触发事件 65
实例 45 多行文本域 66
实例 46 TextArea 触发事件 67
实例 47 画红点 68
实例 48 渐显文字 70
小结 72

第 5 章 Java Swing 图形界面编程 73

实例 49 鼠标画线 73
实例 50 使用画框 75
实例 51 JButton 示例 76
实例 52 JScrollPane 示例 78
实例 53 JCheckBox 实例 80
实例 54 JList 示例 82
实例 55 JTree 示例 84
实例 56 JTable 示例 88
实例 57 Java Application 启动画面制作 91

实例 58 带进度条的启动窗口 92
实例 59 文件选择对话框 95
实例 60 菜单 95
实例 61 对话框 97
实例 62 计算器 102
实例 63 TicTacToe 游戏 107
实例 64 文本编辑器 109
小结 112

第 6 章 Java 多媒体与图形图像处理编程 113

实例 65 画图 113
实例 66 填充图形 116
实例 67 动态填充 118
实例 68 字体 119
实例 69 数字时钟 121
实例 70 纹理与色彩 124
实例 71 模拟时钟 125
实例 72 椭圆形按钮 129
实例 73 裁剪效果 131
实例 74 Applet 音频播放器 135
实例 75 音频播放 139
小结 141

第 7 章 Java 网络编程 142

实例 76 访问并获得 URL 资源 142
实例 77 UDP 传送 143
实例 78 代理服务器 146
实例 79 网页浏览器 147
实例 80 Telnet 149
实例 81 扫描应用端口 150
实例 82 聊天工具 151
实例 83 上传文件 154
实例 84 IP 多组传播 159
实例 85 邮件发送 163
实例 86 FTP 浏览器 165
小结 173

第 8 章 Java 数据库编程 174

实例 87 JDBC-ODBC 桥 174
实例 88 数据库的常见操作 175
实例 89 使用 JDBC 驱动程序连接 SQL Server 178
实例 90 显示数据库的内容 180
实例 91 数据库查询和修改 183
实例 92 事务使用 187
实例 93 设置保存点 188
小结 190

第 9 章 Java B/S 结构编程 191

实例 94 HelloWorld 191
实例 95 读写文件 192
实例 96 网站计数器 193
实例 97 汉字处理 194
实例 98 文件的上传 196
实例 99 获取客户机浏览器的信息 199
实例 100 收发邮件 202
实例 101 JSP 连接数据库 206
实例 102 用 JavaBean 连接数据库 207
实例 103 网络定餐 209
实例 104 简单的 Servlet 程序 218
实例 105 在 Servlet 中实时地创建图像 220
实例 106 简单购物车 221
实例 107 用户注册登录系统 224
小结 235

第 10 章 Java 的 Web Service 编程 236

实例 108 读写 XML 文件 236
实例 109 DOM 树 239
实例 110 web.xml 中的 Servlet 配置 242
实例 111 简单的 SOAP 调用 245
实例 112 访问 SOAP RPC 服务 248
实例 113 查询 UDDI 信息 249
小结 251

第 11 章 EJB 编程 252

实例 114 JNDI 的使用 252

实例 115 构建一个简单的 EJB 253
实例 116 无状态会话 Bean 256
实例 117 简单的有状态会话 Bean 260
实例 118 银行存取 BMP 263
实例 119 MessageDrivenBean 271
实例 120 JSP 调用 EJB 274
小结 276

第 12 章 Java 安全性编程 277

实例 121 一个简单的加密和解密程序——凯撒密码 277
实例 122 DES 加密和解密 278
实例 123 RSA 加密和解密 279
实例 124 数字签名 281
实例 125 创建数字证书 283
实例 126 简单的 SSL 通信 284
实例 127 HTTPS 服务 286
实例 128 密码输入框 289
小结 290

第 13 章 J2ME 编程 292

实例 129 Hello World! 292
实例 130 Command 对象 295
实例 131 查看内存利用 296
实例 132 菜单设计 298
实例 133 记录存储 303
实例 134 使用定时器 308
实例 135 无线 HTTP 连接 310
小结 312

第 14 章 Java 其他编程 313

实例 136 JNI 调用本地 C 语言 313
实例 137 获取本机的 Mac 地址 314
实例 138 获取 Java 的系统信息 315
实例 139 贪食蛇游戏 316
小结 323

第 15 章 Java 工具和服务器配置与集成 324

实例 140 J2SE SDK 的安装和配置 324

实例 141 使用第三方包 325
实例 142 Apache 安装和配置 326
实例 143 Tomcat 的安装和配置 328
实例 144 整合 Tomcat 和 Apache 329
实例 145 整合 Tomcat 和 IIS 331
实例 146 WebLogic 的安装和配置 332
实例 147 整合 Jbuilder 与 WebLogic 333
实例 148 Jboss 的安装与配置 334
实例 149 Eclipse 开发 J2EE 配置 335
实例 150 安装、配置和执行 MIDlet 337
小结 337

参考文献 338

第 1 章　Java 编程初步

本章通过 25 个实例介绍了 Java 编程的基础知识，包括几种数据类型 Integer、Character、String、StringBuffer 的使用，类、继承、动态绑定、接口等面向对象编程的基础，还有 LinkedList、Vector、Properties、ArrayList 等 Java 的数据结构的编程。

实例 1　Hello world！

1. Java 的环境配置

Java 开发工具（JDK）是一个由 Sun 公司开发的免费软件包。这个软件包包括了所有编写 Java 程序和运行 Java 程序所需要的工具。它包括构成 Java 环境的所有基本组件，包括 Java 编译器、Java 解释器、小应用程序浏览器（不用打开兼容 Java 的网页浏览器就可以观察小应用程序）以及一些用于开发 Java 应用程序的程序。

JDK 可以使用户在编写程序时所要做的工作尽可能地减少。许多设计师可能已经习惯了使用集成的开发环境，但它不是集成的开发环境，它没有完全包括开发 Java 程序的所有工具，JDK 的工具是命令行方式的，它没有漂亮的图形用户界面，如 Visual C++6.0 或 Borland 的 C++builder，JDK 工具是在命令提示符下使用的（就像 Windows 95 和 Windows NT 里的 DOS 窗口那样）。JDK 包含源代码的文件就是用文本编辑器编辑的普通 ASCII 码文本文件，这种文本编辑器可以是 NotPad（Win32 系统提供）、vi（UNIX 系统提供）或者 BBEdit（Macintosh 提供）。

本实例介绍了在不用任何 Java 编程工具的情况下只用 JDK 和一般的文本编辑工具完成 Java 程序的编写和执行。先在 Sun 公司的主页（http://java.sun.com）上下载 JDK，有多个版本，选择 Java 2 SDK se1.4.2 版本下载并安装。

1）Path 环境变量的配置

JDK 安装完毕后，在 Windows XP 下进行配置。

（1）在“我的电脑”上单击右键进入“系统属性”，单击“高级”标签项，再选择“环境变量”，在“系统变量(S)”下查找 Path 环境变量，找到后选择“编辑”。

（2）在“变量值”的文本框中，将光标移动到最前方，输入安装的 J2SDK1.4.2 所在的目录下 bin 目录的位置，例如：笔者在机器上安装的是 J2SDK1.4.0，安装到了 D 盘上，其 bin 目录的完整路径为 D:\j2sdk1.4.2\bin，注意加上“;”作为分隔符号。如果在“系统变量(S)”下没有找到 Path 环境变量，那么就选择“新建”。

（3）在“变量名”中输入“Path”，在变量值中输入安装的 J2SDK1.4.2 所在的目录下 bin 目录的位置，然后单击“确定”按钮。

2）CLASSPATH 的配置

在“系统变量(S)”下，选择“新建”，在“变量名”的文本框中输入“CLASSPATH”，在变量值中输入“.”，单击“确定”按钮。

最后在“环境变量”对话框中单击“确定”按钮、结束“Path”和“CLASSPATH”环境变量的配置。

2. Hello world 程序的实现

先建立一个文本文件 HelloWorld.java（仔细检查大小写，Java 是大小写敏感的），然后开始编写这个简单的程序，程序代码如下：

```
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello,World!");
    }
}
```

在 DOS 命令提示符下进入 HelloWorld.java 文件的目录，然后输入 javac HelloWorld.java。机器应该暂停一会儿，然后退回到提示符下。如果出现“javac 找不到”这种错误，是因为 PATH 变量中没有设置 JAVA/BIN 目录。那么就应该回到前面检查一下 JDK 环境变量配置。如果未出现任何错误，就可以开始运行用户的第一个 Java 程序了。在命令提示符下输入下列内容：

```
java HelloWorld
```

这时用户将会看到下列结果：

```
Hello,World!
```

Java 环境通过在编译器和计算机之间增加一个名为 Java 虚拟机（JVM）的中介来解决跨平台的问题。Java 编译器 javac 不是直接面向某种计算机类型的计算机编译器，而是将文本文件中人可读懂的 Java 源代码编译成 Java 虚拟机能够理解的低级 Java 字节码。Java 虚拟机接管这些字节码并对它解释，这样 Java 程序就可以在装有 Java 虚拟机的计算机上运行了。

Java 开发工具包（JDK）中包含以下命令行工具：

javac：基于 Java 编程语言的编译器。

java：Java 应用程序的启动器。

jar：管理 Java 归档（JAR）文件。

javadoc：提取代码中的注释，并根据这些注释生成 HTML 文档。

appletviewer：无需浏览器运行和调试 applet。

实例 2 Integer 类

本实例演示 Integer 类的使用方法。

【编程要点】

Integer 类在某对象中打包了原始类型为 int 的值。Integer 类型对象包含 int 型的单个域。此外，此类提供了许多方法，可以将 int 型转换为 String 型，也可以将 String 型转换为 int 型，还包含处理 int 类型时的其他有用常量和方法。

【程序实现】

IntegerDemo.java 文件内容如下：

```
public class IntegerDemo{
    public IntegerDemo(){
    }
    public static void main(String[] args){
        Integer[] array = {new Integer(20),new Integer(40), new Integer("110")};
        for(int i=0; i<array.length; i++){
            System.out.print(Integer.toBinaryString(array[i].intValue())+"\t");
            System.out.print(Integer.toHexString(array[i].intValue())+"\t");
            System.out.print(Integer.toOctalString(array[i].intValue())+"\n");
```

```
        }
    }
}
```

【运行结果】

```
10100    14   24
101000   28   50
1101110  6e   156
```

【分析与总结】

整数的不同进制的表示方法如下：

toBinaryString(int i)：创建一个整数参数以 2 为基数的无符号整数的字符串表示。

toHexString(int i)：创建一个整数参数的以 16 为基数的无符号整数的字符串表示。

toOctalString(int i)：创建一个整数参数以 8 为基数的无符号整数的字符串表示。

实例 3 显示最大值

本实例显示 Java 常见数据类型的最大值。

【编程要点】

Java 的各种数据类型 int、float、double、byte 等在 java.lang 包里都有各自的类分别和它们对应。其中每个类都有个属性 MAX_VALUE 表示这种类型数据的最大值。

【程序实现】

MaxVariables.java 文件内容如下：

```
public class MaxVariables {                          //java 中的一切都可以用类表示
    public static void main(String args[]) {        //每个程序必须有 main()函数
        //整数类型
        byte largestByte = Byte.MAX_VALUE;           //定义一个 byte 类型的变量
        short largestShort = Short.MAX_VALUE;        //定义一个 short 类型的变量
        int largestInteger = Integer.MAX_VALUE;      //定义一个 int 类型的变量
        long largestLong = Long.MAX_VALUE;           //定义一个 long 类型的变量
        //实数类型
        float largestFloat = Float.MAX_VALUE;        //定义一个 float 类型的变量
        double largestDouble = Double.MAX_VALUE;     //定义一个 double 类型的变量
        //其他基本类型
        char aChar = 'S';                            //定义一个字符
        boolean aBoolean = true;                     //定义一个布尔类型的变量
        //在屏幕上显示对应类型的最大值
        System.out.println("最大的 byte 值是: " + largestByte);
        System.out.println("最大的 short 值是: " + largestShort);
        System.out.println("最大的 integer 值是: " + largestInteger);
        System.out.println("最大的 long 值是: " + largestLong);
        System.out.println("最大的 float 值是: " + largestFloat);
        System.out.println("最大的 double 值是: " + largestDouble);
        if (Character.isUpperCase(aChar)) {          //判断字符是否大写
            System.out.println("字符" + aChar + "是大写的字符");
        } else {
            System.out.println("字符" + aChar + "是小写的字符");
        }
            System.out.println("布尔型变量的值是: " + aBoolean);
    }
}
```

【运行结果】

```
最大的 byte 值是: 127
```

```
最大的 short 值是：32767
最大的 integer 值是：2147483647
最大的 long 值是：9223372036854775807
最大的 float 值是：3.4028235E38
最大的 double 值是：1.7976931348623157E308
字符 S 是大写的字符
布尔型变量的值是：true
```

【分析与总结】

程序中还用到 Character 类的 isUpperCase 方法，用于判断字符是不是大写。

实例 4　计算某年某月的天数

本实例实现计算某年指定月份的天数。

【编程要点】

月份可分为大月和小月，而二月份的天数跟是否闰年有关，所以对二月份要特殊处理。

【程序实现】

DayShower.java 文件内容如下：

```
class DayShower {  //定义一个 DayShower 类
    //本程序的 main()方法
    public static void main(String[] arguments) { //定义 一个 String 对象的数组，用来存
放参数
        //初始化变量
        int yearIn = 2004;          //定义一个整数类型的变量，并且赋初值
        int monthIn = 2;            //定义一个整数类型的变量，并且赋初值
        //检验命令行参数的数量
        if (arguments.length > 0)
            monthIn = Integer.parseInt(arguments[0]);
        if (arguments.length > 1)
            yearIn = Integer.parseInt(arguments[1]);
        //在屏幕上输出得到的结果
        System.out.println(yearIn + "年" + monthIn + "月有"
                + countDays(monthIn, yearIn) + "天");
    }
    //countDays()方法，可以被 DayShower 调用
    static int countDays(int month, int year) { //用 switch 语句来计算某个月的天数
        int count = -1;
        switch (month) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            count = 31;
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            count = 30;
            break;
        case 2:
            if (year % 4 == 0)
                count = 29;
            else
                count = 28;
            if ((year % 100 == 0) & (year % 400 != 0))
```

```
                count = 28;
            }
            return count;//返回count的值
        }
    }
```

【运行结果】

```
2004年2月有29天
```

【分析与总结】

程序中处理命令行参数时用到了Integer类的一个方法：

```
pubic static int parseInt(String s) throws NumberFormatException
```

该方法将字符串参数作为带符号十进制整数来分析。除第一个字符为ASCII字符中减号'-'表示的负数，字符串中的字符都必须是十进制数。若该串不包含一个可分析的整数，则抛出NumberFormatException异常。

从本例我们应该掌握条件语句switch case的使用。

实例5 数值运算

本实例演示如何利用Java的API进行数值运算。

【编程要点】

java.lang包的Math类包含基本数学操作，如指数、对数、平方根和三角函数。

【程序实现】

MathDemo.java文件内容如下：

```
//演示如何调用Java API完成数值运算
public class MathDemo {
    public static void main(String[] args) {
        //演示求绝对值和求最大、最小值函数的用法
        System.out.println("abs(-5) = " + Math.abs(-5));
        System.out.println("max(2.72, 3.14) = " + Math.max(2.72, 3.14));
        System.out.println("min(256, 285) = " + Math.min(256, 285));
        //演示四舍五入函数的用法
        System.out.println("round(3.8) = " + Math.round(3.8));
        System.out.println("round(-3.8) = " + Math.round(-3.8));
        //演示求平方根和求幂函数的用法
        System.out.println("sqrt(2) = " + Math.sqrt(2));
        System.out.println("pow((1+2.25/100), 5) = " + Math.pow((1 + 2.25/ 100), 5));
        //演示指数与对数函数的用法
        System.out.println("E = " + Math.E);
        System.out.println("exp(2) = " + Math.exp(2));
        System.out.println("log(2) = " + Math.log(2));
        //演示天花板与地板函数的用法
        System.out.println("ceil(3.14) = " + (int) Math.ceil(3.14));
        System.out.println("floor(3.14) = " + (int) Math.floor(3.14));
        //演示三角函数的用法
        System.out.println("Pi = " + Math.PI);
        System.out.println("sin(Pi / 2) = " + Math.sin(Math.PI / 2));
        System.out.println("cos(0) = " + Math.cos(0));
    }
}
```

【运行结果】

```
abs(-5) = 5
max(2.72, 3.14) = 3.14
min(256, 285) = 256
round(3.8) = 4
round(-3.8) = -4
sqrt(2) = 1.4142135623730951
pow((1+2.25/100), 5) = 1.1176776934618162
E = 2.718281828459045
```

```
exp(2) = 7.38905609893065
log(2) = 0.693147180559453
ceil(3.14) = 4
floor(3.14) = 3
Pi = 3.141592653589793
sin(Pi / 2) = 1.0
cos(0) = 1.0
```

【分析与总结】

由本例可以了解如何进行常用的数值运算。

实例 6 日历表

本实例演示 util 包中 Calendar 类的使用。

【编程要点】

Calendar 类是一种日期处理类，可以想象它是一个挂在墙壁上的典型日历，有许多日期和页数可以翻阅。

（1）Calendar 类的变量域的概念。每个类元素都是域，并且这些域在 Calendar 类中表现为静态变量。这些变量域，可以通过 get/set 类方法来获得或者设置域值。

```
//获得默认的 Calendar 实例，给它设置时间
Calendarcal = Calendar.getInstance();
intyear = cal.get(Calendar.YEAR);
cal.set(Calendar.MONTH,Calendar.NOVEMBER);
```

（2）Calendar 类的 add 和 roll 方法提供在日期之间转换的能力。每个方法都由一个参数变量和一个参数值来修改，通过这个可为正数或负数的参数值来修改它。仅仅不同的是，add 方法可以向高阶的变量域溢出。具体如下：

```
public abstract void add(int field, int amount)
public abstract void roll(int field, boolean up)
```

（3）日期的计算功能。按照日历的规则，将指定（带符号的）数量的时间添加到给定的时间域。例如，从日历的当前时间减 5，可调用：

```
add(Calendar.DATE, -5)
```

（4）时间域滚动功能。在给定的时间域（向上/向下）滚动一个单个的时间单元。例如，为了将当前日期向上滚动一天，可调用：roll(Calendar.DATE, true)。当在年或 Calendar.YEAR 域滚动时，年值将在 1 和调用 getMaximum(Calendar.YEAR)的返回值之间滚动。当在月或 Calendar.MONTH 域滚动时，其他的域（例如日期）可能会发生冲突需要改变。例如，将日期 01/31/96 滚动一月结果是 03/02/96。当在小时域或 Calendar.HOUR_OF_DAY 域滚动，小时值将在范围 0 到 23 之间滚动，它以 0 开始。

【程序实现】

CalendarDemo.java 文件内容如下：

```
import java.util.*;
public class CalendarDemo {
    public static void main(String args[]) {
        Calendar calendar = Calendar.getInstance(); //创建一个日历对象。
        calendar.setTime(new Date());  //用当前时间初始化日历时间。
        String year = String.valueOf(calendar.get(Calendar.YEAR));
        String month = String.valueOf(calendar.get(Calendar.MONTH) + 1);
        String date = String.valueOf(calendar.get(Calendar.DAY_OF_MONTH));
        String day = String.valueOf(calendar.get(Calendar.DAY_OF_WEEK) - 1);
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);
        int second = calendar.get(Calendar.SECOND);
        System.out.println("现在的时间是：");
```

```
            System.out.println("" + year + "年" + month + "月" + date+"日 " + "星期" + day);
            System.out.println("" + hour + "时" + minute + "分" + second + "秒");
            calendar.set(1949, 9, 1);       //将日历翻到 1949 年 10 月 1 日,注意 9 表示十月
            //返回当前时间，作为从开始时间的 UTC 毫秒值
            long time1949 = calendar.getTimeInMillis();
            calendar.set(2004, 9, 1);       //将日历翻到 2004 年 10 月 1 日，9 表示十月
            //返回当前时间，作为从开始时间的 UTC 毫秒值
            long time2004 = calendar.getTimeInMillis();
            long interdays = (time2004 - time1949) / (1000 * 60 * 60 * 24);
            System.out.println("2004 年 10 月 1 日和 1949 年 10 月 1 日相隔"+interdays+"天");
        }
}
```

【运行结果】

现在的时间是：
2004 年 11 月 2 日 星期 2
13 时 41 分 3 秒
2004 年 10 月 1 日和 1949 年 10 月 1 日相隔 20089 天

【分析与总结】

还有一个隐藏在最通用的 Calendar 的子类中的功能性方法：isLeapYear（判断是否为闰年）方法。

```
Calendar cal = Calendar.getInstance();
booleanleapYear = ( (GregorianCalendar)cal ).isLeapYear(2002);
//这个值是 false
```

尽管它是一个实例方法，isLeapYear 方法的行为表现像静态方法，需要提供年份的参数传值给日历。

实例 7　Character 类

本实例演示 Character 类的使用。

【编程要点】

Character 类是字符数据的对象包装类。Character 对象包含单一的 char 型数据。另外，该类还提供了几种方法，可以用来判断字符类型和实现字符的大小写转换。

Character 类中的许多方法采用“Unicode 属性表”中的术语定义，此属性表为每个已定义的 Unicode 代码指定了一个名称。该表也包括了其他一些属性：诸如十进制数，等价的大写值、小写值，和/或标题值。用户能从以下地址文件中获取该表：

ftp://unicode.org/pub/MappingTables/UnicodeData1.1.5.txt

【程序实现】

CharacterDemo.java 文件内容如下：

```
public class CharacterDemo{
    public CharacterDemo(){
    }
    public static void main(String[] args){
        Character c1 = new Character('A');
        Character c2 = new Character('b');
        //输出字符的 ASCII 码值
        System.out.println(c1.charValue() + 0);
        System.out.println(c2.charValue() + 0);
        //比较两个字符 ASSII 码值大小
        int i = c1.compareTo(c2);
        System.out.println(i);
        if(i==0){
            System.out.println("c1 is equal c2!");
        }
```

```
            else if(i<0){
                System.out.println("c1 is smaller than c2!");
            }
            //大小写转换
            System.out.println(Character.toLowerCase('A')+" "
                    + Character.toUpperCase('a'));
            //判断是否为数字
            System.out.println(Character.isDigit('9'));
        }
}
```

【运行结果】

```
A
98
-33
c1 is smaller than c2!
a A
true
```

【分析与总结】

注意 System.out.println(c1.charValue() + 0)这个语句后面加 0 是为了显示码值，因为 charValue 方法返回的是 char 值，所以加 0 后会自动进行类型转换，输出 ASCII 码值，否则输出的是一个字符。

实例 8 String 类

本实例演示字符串的使用方法。

【编程要点】

String 类表示字符串。在 Java 程序中所有的字符串常量（如“abc”）都被实现为这个类的实例。字符串是常量，它们的值在生成后不能改变。字符串缓冲（StringBuffer）支持可变的字符串。因为字符串对象是不可变的，所以它们可以被共享。

String 类包括检查字符串序列中单个字符的方法，这些方法用于比较字符串、查找字符串、抽取子字符串、生成一个字符串拷贝和把所有字符都转换为大写或小写。Java 语言提供对字符串连接操作符（+）和其他字符串对象相互转换的特殊支持。字符串连结是通过 StringBuffer 类和它的 append 方法实现的。字符串转换是通过 toString 方法实现的，该方法由 Object 定义并且被 Java 中的所有类继承。

【程序实现】

StringDemo.java 文件内容如下：

```
//演示字符串对象的用法
public class StringDemo {
    public static void main(String[] args) {
        //访问字符串长度
        String myName = "G. Leeman";
        int length = "张 三丰".length();
        System.out.println("长度 1: " + myName.length()); //输出 9
        System.out.println("长度 2: " + length);          //输出 3 而不是 6
        //比较字符串大小
        String name1 = "programming in java";
        String name2 = "Programming in Java";
        System.out.println("比较 1: " + name1.equals(name2)); //输出 false
        System.out.println("比较 2: " + name1.equalsIgnoreCase(name2));//输出 true
        System.out.println("比较 3: " + name2.compareTo("Program"));   //输出正数
        //访问字符串中的字符
        System.out.println("字符 1: " + name1.charAt(4));          //输出 r 而不是 g
```

```
            System.out.println("字符2: " + name1.indexOf('a'));        //输出5
            System.out.println("字符3: " + name2.lastIndexOf('a'));    //输出18
            //访问字符串中的子串
            String subname = "in";
            System.out.println("子串1: " + name1.substring(3, 10));    //输出grammin
            System.out.println("子串2: " + "abc".concat("123"));       //输出abc123
            System.out.println("子串3: " + name2.startsWith("Pro"));       //输出true
            System.out.println("子串4: " + name2.endsWith("in Java"));     //输出true
            System.out.println("子串5: " + name1.indexOf(subname));        //输出8
            System.out.println("子串6: " + name1.lastIndexOf(subname));    //输出12
            //字符串的其他操作
            System.out.println("小写: "+name2.toLowerCase()); //输出programming in java
            System.out.println("大写: "+name2.toUpperCase()); //输出PROGRAMMING IN JAVA
            System.out.println("替换: "+name1.replace('a','A'));  //输出 progrAmming in
jAvA
        }
    }
```

【运行结果】

```
长度1: 9
长度2: 3
比较1: false
比较2: true
比较3: 12
字符1: r
字符2: 5
字符3: 18
子串1: grammin
子串2: abc123
子串3: true
子串4: true
子串5: 8
子串6: 12
小写: programming in java
大写: PROGRAMMING IN JAVA
替换: progrAmming in jAvA
```

【分析与总结】

程序演示了String类的常用方法。

实例9 StirngBuffer类

本实例演示StringBuffer类的使用。

【编程要点】

该类提供了对字符串进行添加、删除和插入等操作的类。它是可变长和可写的动态字符序列，它可以根据相应的操作自动地增加空间。

在StringBuffer上的基本操作是append和insert方法，它们都被覆盖以接受任意类型的数据。每个方法都迅速地把所给的数据转换成一个字符串，并且把该字符串的字符添加或插入到字符串缓冲区中。append方法常常把这些字符添加到缓冲区的末端；而insert方法则把这些字符添加到指定的地方。

【程序实现】

StringBufferDemo.java文件内容如下：

```
public class StringBufferDemo{
    public StringBufferDemo(){
    }
    public static void main(String[] args){
        StringBuffer sb = new StringBuffer("Java");
        double d = 1.4;
        sb.append(d).append("have").append(
                new String(" a lot of new features"));
        System.out.println(sb);
        sb.insert(4, "SDK");
        System.out.println(sb);
        sb.delete(4, 10);
        System.out.println(sb.reverse());
    }
}
```

【运行结果】

```
Java1.4have a lot of new features
JavaSDK1.4have a lot of new features
serutaef wen fo tol a evahavaJ
```

【分析与总结】

程序 delete 方法删除 StringBuffer 对象中索引值从 start 到 end 的字符，该方法返回指向该 StringBuffer 对象的引用。

reverse 方法返回一个 StringBuffer 对象，该对象中的字符按原 StringBuffer 对象中字符的相反顺序排列。

实例 10 数组

本实例通过打印正方形的整数矩阵来展示数组的应用。

【编程要点】

数组声明的格式如下：

```
type[] arrayname;
```

或

```
type arrayname[];
```

该格式中 type 的作用是说明数组中各元素的数据类型。[]表示声明的是数组，多维的话用多个[]表示，以区别一般的变量。arrayname 是被声明的数组名字，可以是由字母开头的字母数字串，该字符串中还可带下划线。

【实例程序】

```
import java.io.*;
public class ArrayPrint
{
    private int N;  //矩阵的维数
    private int[][] array;
    ArrayPrint(){
        N=5;
        array=new int[N][N];
    }
    ArrayPrint(int n){
        N=n;
        array=new int[N][N];
    }
    public void Print(){
        int i=0,j=0,i0=0,j0=0,count=1,Num=N-1;
        while(Num>0) {
            for (i=i0,j=j0;j<Num;j++,count++) {
                array[i][j]=count;
            }
            for (i=i0;i<Num;i++,count++) {
                array[i][j]=count;
```

```
            }
            for (j=Num;j>j0;j--,count++) {
                array[i][j]=count;
            }
            for (i=Num;i>i0;i--,count++) {
                array[i][j]=count;
            }
            i0++;
            j0++;
            Num--;
        }
        if (N%2!=0) {  //矩阵的维数为奇数时，给矩阵中间的位置赋值
            array[N/2][N/2]=count;
        }
        for (i=0;i<N;i++) {
            for (j=0;j<N;j++) {
                String ar="0";
                if (array[i][j]/10<1) ar="  "+array[i][j];
                else if (array[i][j]/100<1 && array[i][j]/10>=1) {
                    ar=" "+array[i][j];
                }
                System.out.print(ar+" ");
            }
            System.out.println();
        }
    }
    public static void main(String args[]) throws IOException{
        int order=0;
        System.out.print("Please input the order of the square:(<10)");
        order = System.in.read();
        order = order-'0';
        ArrayPrint arr=new ArrayPrint(order);
        arr.Print();
    }
}
```

【运行结果】

```
Please input the order of the square(<10):
```

输入一个小于10的正整数：5

输出：

```
1    2    3    4   5
16   17   18   19  6
15   24   25   20  7
14   23   22   21  8
13   12   11   10  9
```

【分析与总结】

在定义数组时，int [][] array = new int[N][N]，其中N不必是一个常量，只要在定义之前，N有初值就可以了。

实例11 类

本实例演示Java类的定义和实现。

【编程要点】

类是Java编程中不可缺少的一部分，Java程序的基本单位是类，Java的类描述了对象的特性，是对象的模板，它把数据和操作数据的程序封装在一起。

类变量的声明格式为：

```
[修改符]<类型><类变量名>
```

类变量名必须是惟一的，但Java程序允许类变量名与某个方法的名字同名。修饰符规定了类变量被访问的权限，即类变量的作用域。Java 中类变量的修饰符包括 static、

public、private、protected、private protected 等。各种情况如下：

（1）用修饰符 static 来声明为静态的，则能被所在类的所有对象访问，即所有对象能对静态的类变量进行不同的操作。

（2）用修饰符 public 声明的类变量称为公用变量，其允许 Java 系统中所有的类访问，即该类变量所属的类以及同一包中其他的类访问。

（3）用修饰符 private 声明的类变量称为专用变量，只允许所属的类访问。

（4）用修饰符 protected 声明的类变量称为保护型变量，该变量允许被其所归属的类、由此类派生出来的子类以及同一包中的其他类访问。

（5）用修饰符 private 和 protected 共同声明的类变量称为专用保护型变量，该变量只允许它归属的类或由此类派生的子类访问。

声明变量时若没加修饰符，则为默认型变量，该变量允许被其归属的类访问，也可被同一类包中的其他类访问。

【实例程序】

```
//ComplexNumber.java
package javabasic;

public class ComplexNumber {
    private double x, y;
    public ComplexNumber(double real, double imaginary) {
        this.x = real;
        this.y = imaginary;
    }
    public double real() { return x; } //返回实部
    public double imaginary() { return y; } //返回虚部
    public double magnitude() { return Math.sqrt(x*x + y*y); } //返回共轭值
    public String toString() { return "(" + x + "," + y + ")"; }
    //两个复数相加
    public static ComplexNumber add(ComplexNumber a, ComplexNumber b) {
        return new ComplexNumber(a.x + b.x, a.y + b.y);
    }
    //this 对象与另一个复数相加
    public ComplexNumber add(ComplexNumber a) {
        return new ComplexNumber(this.x + a.x, this.y+a.y);
    }
    //两个复数相乘
    public static ComplexNumber multiply(ComplexNumber a, ComplexNumber b) {
        return new ComplexNumber(a.x*b.x - a.y*b.y, a.x*b.y + a.y*b.x);
    }
    //this 对象与另外一个复数相乘
    public ComplexNumber multiply(ComplexNumber a) {
        return new ComplexNumber(x*a.x - y*a.y, x*a.y + y*a.x);
    }
}
```

【分析与总结】

ComplexNumber 类实现了复数的封装，使得复数之间的相加、相乘等操作只需要调用类中的 add()和 multiply()就能实现。

实例 12 Class 类

本实例演示 Class 类的使用。

【编程要点】

Class 类实例表示当前运行的 Java 应用程序中的类和接口。每个数组均属于一个 Class

类对象，所有具有相同元素类型和维数的数组共享一个 Class 对象。基本的 Java 类型（boolean，byte，char，short，int，long，float 和 double）和 void 类型也可表示为 Class 对象。

对于 Class 类，没有公共构造函数。当各类被调入时，由 Java 虚拟机自动创建 Class 对象，或通过类装载器中的 defineClass 方法生成。

Class 类常用的方法有：

public static native Class forName(String className) throws ClassNotFoundException：返回给定的字符串名对应的 Class 对象。若给定一个类或接口的完整路径名，那么此方法将试图定位、装载和连接该类。若成功，返回该类对象。否则，抛出 ClassNotFoundException 异常。

public native String getName()：返回 Class 对象表示的类型（类、接口、数组或基类型）的完整路径名串。

public native Class getComponentType()：如果此类表示一个数组，则返回表示该数组组件的 Class 对象，否则返回 null。

public Field[] getFields() throws SecurityException：返回一个容纳 Field 对象的数组，该数组表示了 Class 代表的类或接口所有的可访问的公有域。如果该类或接口没有可访问的属性，或者它为数组类或基类型类，则返回长度为零的数组。特殊地，如果此对象是类，则返回该类和其所有父类的公有域。如果是一接口，则返回该接口和其所有父类的域。如果是数组或基类型，则返回一长度为 0 的数组。此方法不影响数组类型的隐含长度域。用户代码应使用数组类的方法处理数组。

public Method[] getMethods() throws SecurityException：返回一个容纳 Method 对象的数组，该数组表示了 Class 代表的类或者接口所有的公有 member 方法，包括已声明的和从父类继承的方法。如果该类或接口没有公有成员方法，则返回长度为零的数组。

public Field getField(String name) throws NoSuchFieldException, SecurityException：返回一个 Field 对象，它表示该 Class 对象代表的类或接口的指定的公有成员域。name 参数是一个 String，它指定所要域的简单名。用给定名，查询该类或接口的所有成员域，返回对应的公有域。

```
public Method getMethod(String name,Class parameterTypes[])
                        throws NoSuchMethodException, SecurityException
```

返回一个 Method 对象，它表示 Class 对象代表的类或接口的指定的公有成员方法。name 参数是一个 String，它指定所要方法的简单名称，parameterTypes 是识别此方法形参类型的 Class 对象的数组，形参以声明的顺序存放在此数组中。用给定名和形参，查询该类或接口的所有成员方法，返回对应的公有方法。如果找不到一个匹配方法则抛出 NoSuchMethodException 的异常，如果拒绝访问此信息则抛出 SecurityException 的异常。

【程序实现】

ClassDemo.java 文件内容如下：

```
public class ClassDemo{

    public ClassDemo(){
    }
    public static void main(String[] args){
        ClassDemo test = new ClassDemo();
        System.out.println("The class of test is " + test.getClass().getName());
```

```
            try{
                Class c = Class.forName("java.lang.Object");
                System.out.println(c.getPackage()+"\t");
                System.out.println(c);
                java.lang.reflect.Method[] m = c.getMethods();
                System.out.println("The element in String[] is "
                        + String[].class.getComponentType());
                System.out.println((new int[3][4][5][6]).getClass().getName());
                System.out.println("The method defined in Object class;");
                for(int i=0; i<m.length; i++)
                    System.out.println(m[i]);
            }
            catch(ClassNotFoundException e){
                e.printStackTrace();
            }
        }
    }
```

【运行结果】

```
The class of test is ClassDemo
package java.lang
class java.lang.Object
The element in String[] is class java.lang.String
[[[[I
The method defined in Object class;
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final void java.lang.Object.wait(long,int) throws
java.lang.InterruptedException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
public final native void java.lang.Object.wait(long) throws
java.lang.InterruptedException
public boolean java.lang.Object.equals(java.lang.Object)
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
public java.lang.String java.lang.Object.toString()
```

【分析与总结】

可以通过 Class 类获取类的其他更多的信息，如通过 getSuperClass()可以返回对象直接父类，通过 isInstance(Object obj)判断对象实例 OBJ 是否为调用该方法的 Class 对象表示的类对象。

实例 13 观察垃圾回收过程

本实例演示垃圾回收过程。

【编程要点】

Java 可用垃圾收集器回收由不再使用的对象占据的内存。现在考虑一种非常特殊且不多见的情况。假定对象分配了一个“特殊”内存区域，没有使用 new。垃圾收集器只知道释放那些由 new 分配的内存，所以不知道如何释放对象的“特殊”内存。为解决这个问题，Java 提供了一个名为 finalize()的方法，可为类定义它。在理想情况下，它的工作原理应该是这样的：一旦垃圾收集器准备好释放对象占用的存储空间，它首先调用 finalize()，而且只有在下一次垃圾收集过程中，才会真正回收对象的内存。所以如果使用 finalize()，就可以在垃圾收集期间进行一些重要的清除或清扫工作。因此 finalize()应该作为一种常规用途的清除方法使用。垃圾收集器存在的惟一原因是为了回收程序不再使用的内存。

例如，假设在对象创建过程中，它会将自己描绘到屏幕上。如不从屏幕明确删除它的图像，那么它可能永远都不会被清除。若在 finalize()里置入某种删除机制，那么假设对象被当作垃圾收掉了，图像首先会将自身从屏幕上移去。但若未被收掉，图像就会保留下来。

【实例程序】

```
//:Garbage.java
//Demonstration of the garbage
//collector and finalization
package javabasic;
class Chair {
  static boolean gcrun = false;
  static boolean f = false;
  static int created = 0;
  static int finalized = 0;
  int i;
  Chair() {
    i = ++created;
    if(created == 47)
      System.out.println("Created 47");
  }
  protected void finalize() {
    if(!gcrun) {
      gcrun = true;
      System.out.println("Beginning to finalize after " +
        created + " Chairs have been created");
    }
    if(i == 47) {
      System.out.println("Finalizing Chair #47, " +
        "Setting flag to stop Chair creation");
      f = true;
    }
    finalized++;
    if(finalized >= created)
      System.out.println("All " + finalized + " finalized");
  }
}

public class Garbage {
  public static void main(String[] args) {
    if(args.length == 0) {
      System.err.println("Usage: \n" +
        "java Garbage before\n  or:\n" + "java Garbage after");
      return;
    }
    while(!Chair.f) {
      new Chair();
      new String("To take up space");
    }
    System.out.println(
      "After all Chairs have been created:\n" +
      "total created = " + Chair.created +
      ", total finalized = " + Chair.finalized);
    if(args[0].equals("before")) {
      System.out.println("gc():");
      System.gc();
      System.out.println("runFinalization():");
      System.runFinalization();
    }
    System.out.println("bye!");
    if(args[0].equals("after"))
      System.runFinalizersOnExit(true);
  }
}
```

【运行结果】

在命令提示符下输入 java Garbage，输出结果为：

```
Usage:
java Garbage before
or:
java Garbage after
```

在命令提示符下输入 java Garbage，输出结果为：

```
Created 47
Beginning to finalize after 5405 Chairs have been created
```

```
Finalizing Chair #47, Setting flag to stop Chair creation
After all Chairs have been created:
total created = 45620, total finalized = 36568
gc():
All 45620 finalized
runFinalization():
bye!
```

在命令提示符下输入 java Garbage，输出结果为：

```
Created 47
Beginning to finalize after 5405 Chairs have been created
Finalizing Chair #47, Setting flag to stop Chair creation
After all Chairs have been created:
total created = 48787, total finalized = 36568
bye!
All 48787 finalized
```

【分析与总结】

上面这个程序创建了许多 Chair 对象，而且在垃圾收集器开始运行后的某些时候，程序会停止创建 Chair。由于垃圾收集器可能在任何时间运行，所以不能准确知道它在何时启动。因此，程序用一个名为 gcrun 的标记来指出垃圾收集器是否已经开始运行。利用第二个标记 f，Chair 可告诉 main()它应停止对象的生成。这两个标记都是在 finalize()内部设置的，它调用于垃圾收集期间。

另两个 static 变量——created 以及 finalized——分别用于跟踪已创建的对象数量以及垃圾收集器已进行完收尾工作的对象数量。最后，每个 Chair 都有它自己的（非 static）int i，所以能跟踪了解它具体的编号是多少。编号为 47 的 Chair 进行完收尾工作后，标记会设为 true，最终结束 Chair 对象的创建过程。

所有这些都在 main()的内部进行，在下面这个循环里：

```
while(!Chair.f) {
new Chair();
new String("To take up space");
}
```

读者可能会疑惑这个循环什么时候会停下来，因为内部没有任何改变 Chair.f 值的语句。然而，finalize()进程会改变这个值，直至最终对编号 47 的对象进行收尾处理。

每次循环过程中创建的 String 对象只是属于额外的垃圾，用于吸引垃圾收集器——一旦垃圾收集器对可用内存的容量感到“紧张不安”，就会开始关注它。

运行这个程序的时候，提供了一个命令行自变量“before”或者“after”。其中，“before”自变量会调用 System.gc()方法（强制执行垃圾收集器），同时还会调用 System.runFinalization()方法，以便进行收尾工作。这些方法都可在 Java 1.0 中使用，但通过使用“after”自变量而调用的 runFinalizersOnExit()方法却只有 Java 1.1 及后续版本提供了对它的支持。注意可在程序执行的任何时候调用这个方法，而且收尾程序的执行与垃圾收集器是否运行是无关的。

在 Java 1.1 中，收尾模块肯定会运行这一许诺已成为现实——但前提是我们明确地强制它采取这一操作。若使用一个不是“before”或“after”的自变量（如“none”），那么两个收尾工作都不会进行，而且我们会得到像下面这样的输出：

```
Created 47
Beginning to finalize after 8694 Chairs have been created
Finalizing Chair #47, Setting flag to stop Chair creation
After all Chairs have been created:
total created = 9834, total finalized = 108
bye!
```

因此，到程序结束的时候，并非所有收尾模块都会得到调用。为强制进行收尾工作，可先调用System.gc()，再调用System.runFinalization()。这样可清除到目前为止没有使用的所有对象。这样做一个稍显奇怪的地方是在调用runFinalization()之前调用gc()，这看起来似乎与Sun公司的文档说明有些抵触，它宣称首先运行收尾模块，再释放存储空间。然而，若在这里首先调用runFinalization()，再调用gc()，收尾模块根本不会执行。

实例14 继承

本实例演示Java中的继承语法，以及它们的构造函数调用顺序。

【编程要点】

Java中的继承是通过extends关键字来实现的，在定义类时使用extends关键字指明新定义类的父类，就在两个类之间建立了继承关系。新定义的类称为子类，它可以从父类哪里继承所有非private的属性和方法作为自己的成员。

【实例程序】

```
//:Cartoon.java
package javabasic;

class Art {
  Art() {
    System.out.println("Art constructor");
  }
}
class Drawing extends Art {
  Drawing() {
    System.out.println("Drawing constructor");
  }
}
public class Cartoon extends Drawing {
  public Cartoon() {
    System.out.println("Cartoon constructor");
  }
  public static void main(String[] args) {
    Cartoon x = new Cartoon();
  }
}
```

【运行结果】

```
Art constructor
Drawing constructor
Cartoon constructor
```

【分析与总结】

从运行结果可以看出Java的继承构造函数是先基类后子类的。

实例15 动态绑定

本实例演示Java的函数调用的动态绑定方式。

【编程要点】

在计算机语言中有两种主要的绑定方式——静态绑定和动态绑定。静态绑定发生于数据结构和数据结构间，程序执行之前，静态绑定发生于编译期，因此不能利用任何运行期的信息，它针对函数调用与函数的主体，或变量与内存中的区块。动态绑定则针对运行期产生的访问请求，只用到运行期的可用信息。

在面向对象的代码中，动态绑定意味着决定哪个方法被调用或哪个属性被访问，将基于这个类本身而不基于访问范围。

【实例程序】

```
//: Shapes.java
package javabasic;
class Shape {
  void draw() { System.out.println(this + ".draw()"); }
}
class Circle extends Shape {
  public String toString() { return "Circle"; }
}
class Square extends Shape {
  public String toString() { return "Square"; }
}
class Triangle extends Shape {
  public String toString() { return "Triangle"; }
}
public class Shapes {
  public static void main(String[] args) {
    //Object 数组
    Object[] shapeList = {
      new Circle(),
      new Square(),
      new Triangle()
    };
    for(int i = 0; i < shapeList.length; i++)
      ((Shape)shapeList[i]).draw();  //需要强制转型
  }
}
```

【运行结果】

```
Circle.draw()
Square.draw()
Triangle.draw()
```

【分析与总结】

可以看出，针对从 Shape 派生出来的所有东西，Shape 建立了一个通用接口——也就是说，所有（几何）形状都可以描绘和删除。派生类覆盖了这些定义，为每种特殊类型的几何形状都提供了独一无二的行为。在循环当中调用 draw 时，编译器会根据不同的 shape 调用不同的 draw，从而达到动态绑定。

实例 16　接口

本实例演示接口的应用。

【编程要点】

interface 不带任何实现细目，它不与任何存储空间相关联。合并多个 interface 对编译器而言也是比较轻松的工作。

【实例程序】

```
//: Adventure.java
package javabasic;
interface CanFight {
  void fight();
}
interface CanSwim {
  void swim();
}
interface CanFly {
  void fly();
}
class ActionCharacter {
  public void fight() {}
}
class Hero extends ActionCharacter
    implements CanFight, CanSwim, CanFly {
```

```
  public void swim() {}
  public void fly() {}
}
public class Adventure {
  public static void t(CanFight x) { x.fight(); }
  public static void u(CanSwim x) { x.swim(); }
  public static void v(CanFly x) { x.fly(); }
  public static void w(ActionCharacter x) { x.fight(); }
  public static void main(String[] args) {
    Hero h = new Hero();
    t(h); //把 h 当作 CanFight
    u(h); //把 h 当作 CanSwim
    v(h); //把 h 当作 CanFly
    w(h); //把 h 当作 ActionCharacter
  }
}
```

【分析与总结】

Hero 结合了 ActionCharacter，以及 CanFight、CanSwim、CanFly 等 interfaces，于是便实现了多重继承。interface 的规则是：可以继承 interface 得到一个 interface，如果要以此新型别生成一个对象，此一型别必须是一个定义齐全的 class。不过虽然 Hero 自身并未提供 fight()的定义，但因 SctionCharacter 拥有 fight()的定义，被自动继承下来。因此编译可以成功通过。

实例 17　异常处理

本实例演示 Java 的异常处理。

【编程要点】

异常（Exception）又称为例外，是特殊的运行错误对象，对应着 Java 语言特定的运行错误处理机制。

Java 中定义了许多异常类，每个异常类都代表了一种运行错误，类中包含了该运行错误的信息和处理错误的方法等内容。每当 Java 程序运行过程中发生一个相应的该异常类的对象，即产生一个异常。一旦一个异常对象产生了，系统中就有相应的机制来处理它，确保不会产生死机、死循环或其他对操作系统的损害，从而保证了整个程序运行的安全性。这就是 Java 的异常处理机制。

【实例程序】

```
//:Throwing.java
package javabasic;
public class Throwing {
  public static void f() throws Exception {
    System.out.println("originating the exception in f()");
    throw new Exception("thrown from f()");
  }
  public static void g() throws Throwable {
    try {
      f();
    } catch(Exception e) {
      System.err.println("Inside g(),e.printStackTrace()");
      e.printStackTrace();
      throw e;
    }
  }
  public static void
  main(String[] args) throws Throwable {
    try {
      g();
    } catch(Exception e) {
```

```
            System.err.println(
              "Caught in main, e.printStackTrace()");
            e.printStackTrace();
          }
        }
      }
```

【运行结果】

```
originating the exception in f()
Inside g(),e.printStackTrace()
java.lang.Exception: thrown from f()
    at javabasic.Throwing.f(Throwing.java:8)
    at javabasic.Throwing.g(Throwing.java:12)
    at javabasic.Throwing.main(Throwing.java:23)
Caught in main, e.printStackTrace()
java.lang.Exception: thrown from f()
    at javabasic.Throwing.f(Throwing.java:8)
    at javabasic.Throwing.g(Throwing.java:12)
    at javabasic.Throwing.main(Throwing.java:23)
```

【分析与总结】

异常的 stack trace 会永远记住真正的发源地——无论它被重置多少次。

实例 18 冒泡排序

本实例实现简单的冒泡排序法。

【编程要点】

最简单的排序方法是冒泡排序方法。这种方法的基本思想是，将待排序的元素看作是竖着排列的“气泡”，较小的元素比较轻，从而要往上浮。在冒泡排序算法中要对这个“气泡”序列处理若干遍。在第一遍处理中，就是自底向上检查一遍这个序列，并时刻注意两个相邻的元素的顺序是否正确。如果中两个相邻元素的顺序不对，即“轻”的元素在下面，就交换它们的位置。显然，处理第一遍之后，“最轻”的元素就浮到了最高位置；处理二遍之后，“次轻”的元素就浮到了次高位置。在作第二遍处理时，由于最高位置上的元素已是“最轻”元素，所以不必检查。一般地，第 i 遍处理时，不必检查第 i 高位置以上的元素，因为经过前面 i-1 遍的处理，它们已正确地排好序。

【程序实现】

```
//SortDemo.java 文件
public class SortDemo {
    public static void main(String[] args) {
        //初始化变量
        int[] arrayOfInts = { 12, 8, 3, 59, 132, 1236, 3400, 8, 352, 37 };
        //循环整个数组
        for (int i = arrayOfInts.length; --i >= 0;) {
            //循环每个数字
            for (int j = 0; j < i; j++) {
                if (arrayOfInts[j] > arrayOfInts[j + 1]) {
                    //将两个数字的位置进行对调
                    int temp = arrayOfInts[j];
                    arrayOfInts[j] = arrayOfInts[j + 1];
                    arrayOfInts[j + 1] = temp;
                }
            }
        }
        //循环整个数组
        for (int i = 0; i < arrayOfInts.length; i++) {
            System.out.print(arrayOfInts[i] + " "); //每个数字后加一个空格
        }
        System.out.println();                          //一行结束
    }
```

```
}
```

【运行结果】

```
3 8 8 12 37 59 132 352 1236 3400
```

【分析与总结】

冒泡排序法有许多改进版，如双向冒泡排序法，其基本思想是先让重的气泡沉到底下，然后让轻的气泡浮上来，然后再让较重气泡沉下去，让较轻气泡浮上来，依次反复，直到排序结束。读者可以自行完成。

实例 19　Hanio 塔问题

本实例用递归方法解决著名的 Hanio 塔问题。

【编程要点】

本程序要解决的问题是将 n 个盘，三根柱，每个盘大小不一，并且由小到大放在一根柱上，现在要把它们移到另外一根柱上，并要求由小到大放。

【程序实现】

HanioTower.java 文件内容：

```
public class HanoiTower {
    //将 n 个盘从 from 柱移到 to 柱，以 aux 柱为辅助柱
    public static void move(int n, char from, char to, char aux) {
        if (n == 1) {
            //仅有一个盘时，直接从 from 柱移到 to 柱
            System.out.println("将#1 盘从 " + from + " 移到 " + to);
        } else {
            //将 n - 1 个盘从 from 柱移到 aux 柱，以 to 柱为辅助柱
            move(n - 1, from, aux, to);
            //将最下的圆盘从 from 柱移到 to 柱
            System.out.println("将#" + n + "盘从 " + from + " 移到 " + to);
            //将 n - 1 个盘从 aux 柱移到 to 柱，以 from 柱为辅助柱
            move(n - 1, aux, to, from);
        }
    }
    public static void main(String[] args) {
        //将 3 个圆盘从 A 柱移到 C 柱，移动时利用 B 柱为辅助柱
        move(3, 'A', 'C', 'B');
    }
}
```

【运行结果】

```
将#1 盘从 A 移到 C
将#2 盘从 A 移到 B
将#1 盘从 C 移到 B
将#3 盘从 A 移到 C
将#1 盘从 B 移到 A
将#2 盘从 B 移到 C
将#1 盘从 A 移到 C
```

【分析与总结】

本例介绍一般递归程序的设计和实现，这里将 n 维的问题转化为 n-1 维的问题，然后利用递归就可以轻易地解决 n 维的问题。

实例 20　链表

本实例实现数据结构中常用的单链表。

【编程要点】

链表是一种重要的数据结构，在程序设计中占有很重要的地位。C 语言和 C++语言中是用指针来实现链表结构的，由于 Java 语言不提供指针，所以有人认为在 Java 语言中不能实现链表，其实不然，Java 语言比 C 和 C++更容易实现链表结构。Java 语言中的对象引用实际上是一个指针（本文中的指针均为概念上的意义，而非语言提供的数据类型），所以我们可以编写这样的类来实现链表中的结点。

```
class Node {
    Object data;
    Node next;   //指向下一个结点
}
```

将数据域定义成 Object 类是因为 Object 类是广义超类，任何类对象都可以给其赋值，增加了代码的通用性。为了使链表可以被访问还需要定义一个表头，表头必须包含指向第一个结点的指针和指向当前结点的指针。为了便于在链表尾部增加结点，还可以增加一指向链表尾部的指针，另外还可以用一个域来表示链表的大小，当调用者想得到链表的大小时，不必遍历整个链表。

【程序实现】

List.java 文件内容如下：

```
import java.io.*;

public class List {
    //用变量来实现表头
    private Node Head = null;
    private Node Tail = null;
    private Node Pointer = null;
    private int Length = 0;
    /** 清空整个链表 */
    public void deleteAll() {
        Head = null;
        Tail = null;
        Pointer = null;
        Length = 0;
    }
    /** 链表复位，使第一个结点成为当前结点 */
    public void reset() {
        Pointer = null;
    }
    /** 判断链表是否为空 */
    public boolean isEmpty() {
        return (Length == 0);
    }
    /** 判断当前结点是否为最后一个结点 */
    public boolean isEnd() {
        if (Length == 0)
            throw new java.lang.NullPointerException();
        else if (Length == 1)
            return true;
        else
            return (cursor() == Tail);
    }
    /** 返回当前结点的下一个结点的值，并使其成为当前结点 */
    public Object nextNode() {
        if (Length == 1)
            throw new java.util.NoSuchElementException();
        else if (Length == 0)
            throw new java.lang.NullPointerException();
        else {
            Node temp = cursor();
            Pointer = temp;
```

```
            if (temp != Tail)
                return (temp.next.data);
            else
                throw new java.util.NoSuchElementException();
        }
    }
    /** 返回当前结点的值 */
    public Object currentNode() {
        Node temp = cursor();
        return temp.data;
    }
    /** 在当前结点前插入一个结点，并使其成为当前结点 */
    public void insert(Object d) {
        Node e = new Node(d);
        if (Length == 0) {
            Tail = e;
            Head = e;
        } else {
            Node temp = cursor();
            e.next = temp;
            if (Pointer == null)
                Head = e;
            else
                Pointer.next = e;
        }
        Length++;
    }

    /** 返回链表的大小 */
    public int size() {
        return (Length);
    }
    /**
     * 将当前结点移出链表，下一个结点成为当前结点，如果移出的结点是最后一个结点，则第一个结点成为当前结点
     */
    public Object remove() {
        Object temp;
        if (Length == 0)
            throw new java.util.NoSuchElementException();
        else if (Length == 1) {
            temp = Head.data;
            deleteAll();
        } else {
            Node cur = cursor();
            temp = cur.data;
            if (cur == Head)
                Head = cur.next;
            else if (cur == Tail) {
                Pointer.next = null;
                Tail = Pointer;
                reset();
            } else
                Pointer.next = cur.next;
            Length--;
        }
        return temp;
    }

    /** 返回当前结点的指针 */
    private Node cursor() {
        if (Head == null)
            throw new java.lang.NullPointerException();
        else if (Pointer == null)
            return Head;
        else
            return Pointer.next;
    }
```

```
    /** 链表的简单应用举例 */
    public static void main(String[] args) {
        List a = new List();
        for (int i = 1; i <= 10; i++)
            a.insert(new Integer(i));
        System.out.println(a.currentNode());
        while (!a.isEnd())
            System.out.println(a.nextNode());
        a.reset();
        while (!a.isEnd()) {
            a.remove();
        }
        a.remove();
        a.reset();
        if (a.isEmpty())
            System.out.println("There is no Node in List \n");
        System.out.println("You can press return to quit\n");
        try {
            //确保用户看清程序运行结果
            System.in.read();
        } catch (IOException e) {
        }
    }
}

//构成链表的结点定义
class Node {
    Object data;
    Node next;
    Node(Object d) {
        data = d;
        next = null;
    }
}
```

【运行结果】

```
10
9
8
7
6
5
4
3
2
1
There is no Node in List
```

【分析与总结】

java.util 包中已经有现成的 Linkedlist 类供用户使用，而且已经实现了比较全面的方法。这里实现单链表目的是为了跟 C++的单链表实现方法做个对比。

实例 21 用 LinkedList 实现栈

本实例演示如何用 LinkedList 实现栈。

【编程要点】

如图 1-1 所示为 Collection 对象的继承图，Collection 的子对象共分三类：List、Set 和 Vector，List 和 Set 最大的分别在于 List 可以存放两个以上同样的对象；Set 则是一种对象只会存放一个，当同一种对象存两次时，也只会存一个；Vector 则可以把它当作动作数组。底下则分别为 LinkedList、ArrayList、HashSet、TreeSet、Vector 及 Stack 的范例程序。要特别注意，Vector 和 Stack 是 Java 1 就有的旧类别，其所有 method 都宣告为 synchronized，以避免当一个以上执行者同时存取时会发生不可预期的结果，其余类别是 Java 1.2 才加入

的，并没有宣告为 synchronized，所以使用在多线程程序设计时，要加上临界区间的处理。

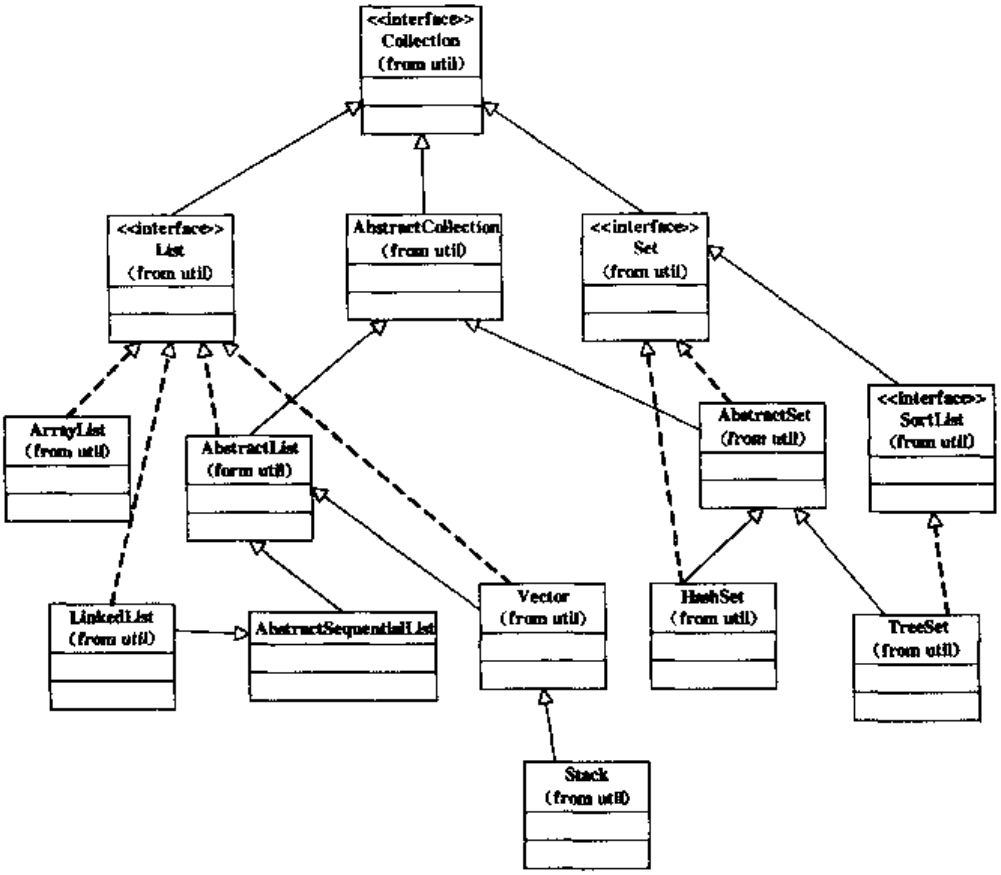


图 1-1

LinkedList 是 Java 提供的链表，它可以透过 listIterator 取得 ListIterator 定位器来存取串行中的元素。

【程序实现】

LinkedListDemo.java 文件内容如下：

```
import java.util.LinkedList;
//很容易使用 LinkedList 实现一个堆栈
class LinkedListStack {
    //堆栈内部使用一个 LinkedList 容器存放元素
    private LinkedList data = new LinkedList();
    public boolean isEmpty() {
        return data.isEmpty();
    }
    public Object top() {
        //取栈顶元素相当于取 LinkedList 容器的头位置的元素
        return data.getFirst();
    }
    public void push(Object element) {
        //压入元素相当于在 LinkedList 容器的头位置插入元素
        data.addFirst(element);
    }
    public void pop() {
        //弹出元素相当于删除 LinkedList 容器的头位置的元素
        data.removeFirst();
    }
}
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedListStack stack = new LinkedListStack();
```

```
            //下面的语句使用类 LinkedList 实现的堆栈的用法
            stack.push("C++");
            stack.push("C#");
            stack.push("Java");
            while (! stack.isEmpty()) {
                System.out.println(stack.top());
                stack.pop();
            }
        }
}
```

【运行结果】

```
Java
C#
C++
```

【分析与总结】

其实 Stack 类已经在 java.util 包中定义，它是继承自 Vector 类的，这里只是演示 LinkedList 的使用。

实例 22 Vector 类

本实例演示 Vector 类的使用方法。

【编程要点】

Vector 类是被广泛应用的一个数据结构类，它实现了可动态扩充的对象数组。类似数组，它包含的元素可通过数组下标来访问。但是，在 Vector 创建之后。Vector 可根据增加和删除元素的需要来扩大或缩小。为了兼容 Java 2 提出的 Collection 数据结构体系，Vector 类在 Java 2 中实现了 List 接口，List 接口扩展于 Collection。因此，Vector 类又具有许多 Collection 接口特性，例如对迭代器（Iterator）的支持，以及通过 Collection 对象来构造 Vector 对象等。

常用方法有以下几种：

public final synchronized void addElement(Object obj)：在向量尾部添加一个指定组件，并把它的长度加一。当向量大小超过它的容量时向量的容量自动增加。

public final synchronized void copyInto(Object anArray[])：把该向量的元素复制到指定数组中。数组必须足够大到可容纳向量中的所有对象。

public final int indexOf(Object elem)：查找给定参数在向量中第一次出现的位置，并用 equals 方法测试它们是否相等。如果该对象未找到，则返回-1。

public final synchronized Object elementAt(int index)：返回指定下标处的组件。如果给出无效的下标则抛出 ArrayIndexOutOfBoundsException。

public final synchronized Object firstElement()：返回该向量的第一个元素。如果该向量没有元素则抛出：NoSuchElementException。

public final synchronized Object lastElement()：返回向量的最后一个元素。如果该向量为空则抛出：NoSuchElementException。

public final synchronized void setElementAt(Object obj,int index)：设置在向量中指定的 index 处的元素为指定的对象。在该位置的先前元素被抛弃。下标值必须大于或等于 0，并且小于向量的当前大小。如果下标无效则抛出 ArrayIndexOutOfBoundsException。

【程序实现】

VectorDemo.java 文件内容如下：

```
import java.util.*;
public class  VectorDemo{
    public VectorDemo(){
    }

    public static void main(String[] args){
        Vector v = new Vector();
        v.add(new Integer(13));
        v.add(new Integer(5));
        v.add(new Integer(8));
        v.add(new Integer(9));
        v.add(new Integer(2));
        //在向量中的元素 9 前插入一个新元素
        v.insertElementAt(new Integer(7), v.indexOf(new Integer(9)));
        //输出向量中的所有元素
        for(int i=0; i<v.size(); i++)
            System.out.print(v.elementAt(i)+"\t");
        System.out.println();
        //输出向量中的第一个元素
        System.out.println("The first element in Vector is " + v.firstElement());
        //输出向量中的最后一个元素
        System.out.println("The last element in Vector is "+ v.lastElement());
        Object[] content = new Object[v.size()];
        v.copyInto(content);
        int[] contentnumber = new int[v.size()];
        for(int i=0; i<content.length; i++)
            contentnumber[i] = Integer.parseInt(content[i].toString());
        //将整型数组中的元素升序排列，并输出
        Arrays.sort(contentnumber);
        for(int i=0; i<contentnumber.length; i++)
            System.out.print(contentnumber[i] +"\t");
    }
}
```

【运行结果】

```
13   5    8    7    9    2
The first element in Vector is 13
The last element in Vector is 2
2    5    7    8    9    13
```

【分析与总结】

Vector 在参数传递中发挥着举足轻重的作用。例如，在界面编程中会碰到这样的情况：在 Applet 中有一块画布（Canvas）和一个 Panel，而 Panel 中放着用户要输入的信息，根据这些信息把参数传递到 canvas 中，这时在 Java 中用一个接口（Interface），而在接口中需用一个 Vector 去传递这些参数。另外，在一个类向另一个类参数传递就可用这种方法。

实例 23　Properties 类

本实例演示 Properties 类的使用。

【编程要点】

Properties 类描述了一个持久的特性集。Properties 可被保存在流中或从流中装入。特性列表中的每个关键字和它的相应值是一个字符串。

一个特性列表可包含另一个特性列表作为它的“缺省”；如果特性关键字没有在原始的特性列表中找到，则搜索第二个特性列表。

【程序实现】

PropertiesDemo.java 文件内容如下：

```
import java.util.*;
```

```
public class PropertiesDemo{
    public PropertiesDemo(){
    }

    public static void main(String[] args){
        PropertiesDemo propertiesDemo = new PropertiesDemo();
        Properties p = new Properties();
        //设置属性
        p.setProperty("C", "China");
        p.setProperty("A", "America");
        p.setProperty("J", "Japan");
        p.setProperty("K", "Korea");
        p.setProperty("S", "Spain");
        //枚举属性
        Enumeration e = p.propertyNames();
        System.out.print("The all keys in p:");
        while(e.hasMoreElements())
            System.out.println(e.nextElement().toString()+"\t");
        System.out.println();
        System.out.println("The property of key 'K' is " + p.getProperty("K"));
        System.out.println("The property of key 'J' is " + p.getProperty("J"));
        System.out.println("The property of key 'Q' is " + p.getProperty("Q"));
    }
}
```

【运行结果】

```
The all keys in p:A
K
J
S
C

The property of key 'K' is Korea
The property of key 'J' is Japan
The property of key 'Q' is null
```

【分析与总结】

主要用到 setProperty 和 getProperty 方法：

public Object setProperty(String key，String value)：将指定值 value 与关键字 key 关联，返回先前与 key 对应的属性值。

public String getProperty(String key, String defaultValue)：返回属性表中指定关键字 key 对应的属性值。若 key 不在属性表中，则返回默认值 defaultValue。

这里用到了 Enumeration 类，一个实现 Enumeration 接口的对象生成一系列元素，每次生成一个。逐次调用 nextElement 方法，逐次返回一系列的元素。

实例 24 ArrayList 类

本实例演示 ArrayList 类的使用。

【编程要点】

ArrayList 类是支持可变长度的动态数组，可以容纳任何对象，包括 null 值。ArrayList 类的行为特征与 Vector 类很相似。

【程序实现】

```
import java.util.*;

public class ArrayListDemo{
    public ArrayListDemo(){
    }
    public static void main(String[] args){
        ArrayList a = new ArrayList();
```

```
        //往 ArrayList 中添加元素
        a.add("a");
        a.add("b");
        a.add("c");
        a.add("d");
        System.out.println(a);
        //获得 Iterator 对象
        Iterator it = a.iterator();
        it.next();
        it.remove();
        System.out.println(a);
        it.next();
        it.remove();
        System.out.println(a);
    }
}
```

【运行结果】

```
[a, b, c, d]
[b, c, d]
[c, d]
```

【分析与总结】

这里主要用到了 ArrayList 的 add 方法和 Iterator 接口的 remove 方法：

boolean add(Object o)：判断该 ArrayList 对象中有无指定对象 o，如果没有就将对象 o 加入到 ArrayList 对象中，并返回 true。如果已经含有该对象则返回 false。

Iterator 接口提供了遍历集合中所有对象的功能。该接口与 Enumeration 接口相似，都定义了判断是否还有元素被迭代或枚举的 hasNext()和 hasMoreElements()方法以及返回下一个迭代或枚举对象的 next()和 nextElement()方法。不同的是，在 Iterator 接口中定义了删除元素的方法。remove 方法删除迭代器中最近一次调用 next()方法返回的元素，该方法必须在调用 next 方法之后调用，否则将抛出 IllegalStateException 异常。

实例 25　线程初探

本实例初步展现 Java 的多线程编程的基本要素。

【编程要点】

产生线程最简单的办法是继承自 Thread 类，这个 class 拥有产生、运行线程的所有必要机制。Thread 最重要的函数就是 run()，必须覆写它，使线程执行用户所指派的工作。因此 run()是程序中会和其他线程“同时”执行的部分。

【实例程序】

```
//SimpleThread.java
package javabasic;

public class SimpleThread extends Thread {
  private int countDown = 5;
  private static int threadCount = 0;
  private int threadNumber = ++threadCount;
  public SimpleThread() {
    System.out.println("Making " + threadNumber);
  }
  public void run() {
    while(true) {
      System.out.println("Thread " +
        threadNumber + "(" + countDown + ")");
      if(--countDown == 0) return;
    }
  }
  public static void main(String[] args) {
```

```
    for(int i = 0; i < 5; i++)
      new SimpleThread().start();
    System.out.println("All Threads Started");
  }
}
```

【运行结果】

```
Making 1
Making 2
Making 3
Making 4
Making 5
All Threads Started
Thread 1(5)
Thread 1(4)
Thread 1(3)
Thread 1(2)
Thread 1(1)
Thread 2(5)
Thread 2(4)
Thread 2(3)
Thread 2(2)
Thread 2(1)
Thread 3(5)
Thread 3(4)
Thread 3(3)
Thread 3(2)
Thread 3(1)
Thread 4(5)
Thread 4(4)
Thread 4(3)
Thread 4(2)
Thread 4(1)
Thread 5(5)
Thread 5(4)
Thread 5(3)
Thread 5(2)
Thread 5(1)
```

【分析与总结】

这个程序产生某个数量的线程，并指派给每个线程一个独一无二的识别号（通过 static 变量产生）。程序覆写了 Thread 的 run()，每次进入循环时，就将计数值减 1，直到计数值为 0 时才结束循环。run()实际上具有某种形式的循环，这种循环会不断运行，直到程序不再需要该线程为止。所以，必须给此一循环设定脱离条件。这意味着除非某个能使 run()终止的外部因素被触发，否则 run()将不断执行下去。

在 main()中，可看到创建并运行了大量线程。Thread 包含了一个特殊的方法，叫作 start()，它的作用是对线程进行特殊的初始化，然后调用 run()。所以整个步骤包括：调用构建器来构建对象，然后用 start()配置线程，再调用 run()。如果不调用 start()——如果适当的话，可在构建器那样做——线程便永远不会启动。可注意到这个例子中并未调用 sleep()，然而输出结果指出每个线程都获得了属于自己的那一部分 CPU 执行时间。从中可以看出，尽管 sleep()依赖一个线程的存在来执行，但却与线程的执行与否无关。它只是函数罢了。main()创建 Thread 对象时，它并未捕获任何一个对象的句柄。普通对象对于垃圾收集来说是一种“公平竞赛”，但线程却并非如此。每个线程都会“注册”自己，所以某处实际存在着对它的一个引用。这样一来，垃圾收集器便只好对它“瞠目以待”了。

小结

本章重点介绍了 Java 语言的数据类型的编程，其中 String、StringBuffer 等类是 Java

编程中经常用到的类型，应该掌握它们的常用方法。

Java作为一种纯面向对象语言，类、接口、动态绑定等面向对象技术都是应该掌握的。

Java的数据结构也是本章的一个重点，Java数据结构的类均定义在java.util包中，其中既包括在JDK1.0中已被广泛应用的Enumeration接口、HashTable类、Properties类、Stack类以及Vector类等，也包括Java 2中新增的Collection接口、Iterator接口、List接口、ListIterator接口、Map接口、AbstractList类、ArrayList类、Collections类和LinkedList类等。熟练掌握这些类对于编写高效简洁的应用程序是必不可少的。

第 2 章　Java 多线程编程

本章介绍 Java 语言的一个重要功能特点，就是对多线程的支持，编程人员可以很方便地开发具有多线程功能、能同时处理多个任务的功能强大的应用程序。本章通过 7 个实例介绍 Java 多线程编程的基础以及应用。另外还编程实现了多线程的著名问题，如：生产者－消费者问题、哲学家就餐问题。

实例 26　简单的多线程应用

本实例是多线程的简单演示。

【编程要点】

本实例所要演示的多线程应用是模拟三个线程要进行大规模的计算（compute()方法），当一个线程调用完 compute()后，休眠一段时间，而另外的某个线程就可以执行。

线程 Tread 是程序中的一个执行流程。Java 虚拟机允许应用程序同时有多个线程在执行。每个线程有一个优先级。高优先级的线程比低优先级线程优先执行。每个线程可能会被标记为后台驻留程序（daemon）也可能不被标记。当线程中运行的程序代码又创建新 Thread 对象时，新线程的初始优先级被置为与其创建线程相同，并且当且仅当其创建线程为后台驻留线程时，它也为后台驻留线程。

当 Java 虚拟机初启时，只有一个 non-daemon 线程（典型地把该方法叫做某种指定类的 main 方法）。Java 虚拟机继续执行线程直到发生下列情况：运行 Runtime 类的方法 exit 被调用并且安全管理程序允许执行退出操作；所有非后台驻留线程已经死亡，例如从方法 run 的调用中退出了或执行了方法 stop。

有两种途径创建新的执行线程。一种是声明 Thread 的子类。该子类必须重载 Thread 类的方法 run。然后分配该子类的实例并启动。创建线程的另一种途径是声明一个类，该类实现 Runnable 接口。然后再实现方法 run。然后可以分配该类的实例，当创建并启动 Thread 时将其作为参数。

【程序实现】

```
//ThreadDemo.java
package thread;

public class ThreadDemo extends Thread {

    //重写 run 方法
    public void run() {
        for (int i = 0; i < 5; i++)
            compute();
    }

    //mian 方法产生另外两个线程和原来解释器的初始线程加在一起共三个一起执行
    public static void main(String[] args) {
        //创建第一个线程
        ThreadDemo thread1 = new ThreadDemo();

        //创建第二个线程
        Thread thread2 = new Thread(new Runnable() {
            public void run() {
```

```
                for (int i = 0; i < 5; i++)
                    compute();
            }
        });
        //设置这两个线程的属性
        if (args.length >= 1)
            thread1.setPriority(Integer.parseInt(args[0]));
        if (args.length >= 2)
            thread2.setPriority(Integer.parseInt(args[1]));
        //启动线程
        thread1.start();
        thread2.start();
        //这个 main()方法是由 Java 解释器产生的初始线程来运行的
        for (int i = 0; i < 5; i++)
            compute();
    }

    //ThreadLocal 对象有一个属性值可以 get 和 set 的，可以维护不同的线程的不同属性值
    //用于记录每个线程调用 compute 方法的次数
    static ThreadLocal numcalls = new ThreadLocal();

    //供线程调用的方法
    static synchronized void compute() {
        //计算被当前线程调用的次数
        Integer n = (Integer) numcalls.get();
        if (n == null)
            n = new Integer(1);
        else
            n = new Integer(n.intValue() + 1);
        numcalls.set(n);

        //显示线程的名称和它的调用次数
        System.out.println(Thread.currentThread().getName() + ": " + n);

        //模拟线程需要大量计算
        for (int i = 0, j = 0; i < 1000000; i++)
            j += i;
        try {
            //线程休眠一段时间
            Thread.sleep((int) (Math.random() * 100 + 1));
        } catch (InterruptedException e) {
        }
     //当前线程暂停并允许其他线程执行，因此不会有饥饿状态的线程
     Thread.yield();
   }
}
```

【运行结果】

```
main: 1
Thread-1: 1
Thread-0: 1
main: 2
Thread-1: 2
Thread-0: 2
main: 3
Thread-1: 3
Thread-0: 3
main: 4
Thread-1: 4
Thread-0: 4
main: 5
Thread-1: 5
Thread-0: 5
```

【分析与总结】

用 synchronized 关键字的方法表示该方法不能被多个线程同时调用。这里用到了

Thread 类的三个方法 start、sleep 和 yield 方法。其中 start 方法使该线程开始执行，Java 虚拟机调用线程 run 方法；run 方法是每个 Thread 的子类都必须覆写它的，如果该线程是用不同的 Runnable 运行对象构造的，则该 Runnable 对象的 run 方法被调用。否则，该方法不做任何事就返回，所以我们如果将 thread2 去掉的话，我们只看到 main Thread 的运行结果。

构造函数 Thread(Runnable target)分配新 Thread 对象。该构造函数与 Thread(null, null,gname)有相同效果，这里 gname 是新生成的名字。自动生成的名字格式为"Thread-"+n 名，这里 n 是整数。

实例 27 简单动画

本实例利用多线程实现简单动画。

【编程要点】

Java 的多线程的程序设计是允许一个程序里的同一时刻中，并发运行各自的线程代码，各线程间操作互不影响。

例如：某些集成开发环境软件，在进行编译过程的同时，还可响应其他操作请求，如继续编辑等，不必等待编译过程完全结束再进行下一步操作。

本实例实现动画的基本思想是在主线程中再产生一个新的线程，由它来专门执行循环操作，实时地控制画面的大小，从而产生动画效果。

【程序实现】

```
//Mutative.java
package thread;
import java.awt.*;
import java.awt.event.*;

public class Mutative {
    public static void main(String args[]) {
        Mywin win = new Mywin();
        win.pack();
    }
}

class Mywin extends Frame implements Runnable {
    Button b = new Button("ok");
    int x = 5;
    Thread thread = null;
    Mywin() {
        setBounds(100, 100, 200, 200);
        setLayout(new FlowLayout());
        setVisible(true);
        add(b);
        b.setBackground(Color.green);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
      //创建一个新的线程，窗口做目标对象，替线程 thread 实现接口 Runnable
        thread = new Thread(this);
      //在创建窗口时又开始了线程 thread
        thread.start();
    }

    public void run()
    {
```

```
            while (true) {
                x = x + 1;
                if (x > 100)
                    x = 5;
                b.setBounds(40, 40, x, x);
                try {
                    thread.sleep(200);
                } catch (InterruptedException e) {
                }
            }
        }
    }
```

【运行结果】

运行结果如图 2-1 所示。

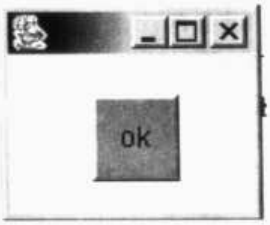


图 2-1

【分析与总结】

本实例是通过实现 Runnable 接口来实现多线程的，因为 Mywin 类已经继承了 Frame 类，所以不能通过继承 Thread 类来实现多线程。在 run 方法中，通过 setBound 来实时地改变按钮的大小。

Java 动画基本上都要采用多线程的机制来实现，读者可以用多线程的机制制作出更多更好看的动画。

实例 28　模拟接力

本实例用多线程演示一个模拟的接力跑。

【编程要点】

本例模拟的接力跑中有一个裁判，两个运动员。第一个运动员跑完一百米将接力棒交给下一个运动员，下一个运动员跑完一百米后结束。这里关键是线程何时开始，何时停止。这里是通过“中断”机制来实现的，当运动员 1 线程完成后，调用 interrupt 方法中断线程，然后程序就可以捕获到 InterruptedException，从而可以开始下一线程。

【程序实现】

```
//Run.java
package thread;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Run extends Applet implements Runnable, ActionListener {
    Button b = new Button("go");
    TextField text = null;
    Thread judgment, athlete1, athlete2;
    int x = 10;                    //线程运动的起始位置。
    Graphics mypen = null;
    public void init() {
        b.addActionListener(this);
        text = new TextField(20);
```

```
        judgment = new Thread(this);
        athlete1 = new Thread(this);
        athlete2 = new Thread(this);
        add(b);
        add(text);
        mypen = getGraphics();
    }
    public void start() {
        judgment.start();
    }
    public void actionPerformed(ActionEvent e) {
        judgment.interrupt(); //单击按钮结束发令员的“生命”
    }
    public void run() {
        if (Thread.currentThread() == judgment) {
            while (true) {
                text.setText("准备跑... ...");
                text.setText("......");
                try {
                    judgment.sleep(30);
                } catch (InterruptedException e) {
                //单击按钮结束“生命”，并让运动员_1 开始跑
                    text.setText("跑");
                    athlete1.start();
                    break;
                }
            }
        }
        if (Thread.currentThread() == athlete1) {
            while (true) {
                x = x + 1;
                mypen.setColor(Color.blue);
                mypen.clearRect(10, 80, 99, 100);        //显示线程运动的动画
                mypen.fillRect(x, 85, 5, 5);
                try {
                    athlete1.sleep(10);
                } catch (InterruptedException e) {
                //通知运动员_2 开始跑，运动员_1 结束“生命”
                    athlete2.start();
                    return;
                }
                if (x >= 100) {
                    athlete1.interrupt();       //运动员_1 当跑到 100 米处时结束“生命”
                }
            }
        }
        if (Thread.currentThread() == athlete2) {
            while (true) {
                x = x + 1;
                mypen.setColor(Color.red);
                mypen.clearRect(105, 80, 150, 100);      //显示线程运动的动画
                mypen.fillRect(x + 5, 85, 7, 7);
                try {
                    athlete2.sleep(10);
                } catch (InterruptedException e) {
                    text.setText("到达终点");
                    return;
                }
                if (x >= 200)  //运动员_2 跑到 200 米处时结束“生命”
                {
                        athlete2.interrupt();
                }
            }
        }
    }
}
```

【运行结果】

运行结果如图 2-2 所示。

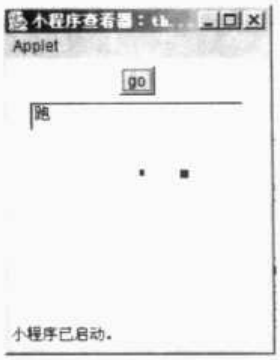


图 2-2

裁判鸣枪用 go 按钮来模拟，单击 go 后，蓝色的方块代表第一个运动员，红色的方块代表第二个运动员，第一个运动员跑到中间位置后第二个运动员接着跑完余下的。

【分析与总结】

从本例可以看出多线程在动画设计中的重要性。

实例 29　线程树

本实例演示如何获取线程树的全面信息。

【编程要点】

线程组是 Java 特有的概念，在 Java 中，线程组是类 ThreadGroup 的对象，每个线程都隶属于惟一一个线程组，这个线程组在线程创建时指定并在线程的整个生命期内都不能更改。可以通过调用包含 ThreadGroup 类型参数的 Thread 类构造函数来指定线程所属的线程组，若没有指定，则线程缺省地隶属于名为 system 的系统线程组。

在 Java 中，除了预建的系统线程组外，所有线程组都必须显式创建。在 Java 中，除系统线程组外的每个线程组又隶属于另一个线程组，可以在创建线程组时指定其所隶属的线程组，若没有指定，则缺省地隶属于系统线程组。一个线程组也可以包含其他线程组。线程组构成一棵树，除了初始的线程组之外都有一个双亲。可允许一个线程存取它自己所在线程组的信息，但是不允许存取它的线程组的父线程组或其他线程组的信息。这样，所有线程组组成了一棵以系统线程组为根的树。

Java 允许对一个线程组中的所有线程同时进行操作，比如可以通过调用线程组的相应方法来设置其中所有线程的优先级，也可以启动或阻塞其中的所有线程。

Java 的线程组机制的另一个重要作用是线程安全。线程组机制允许我们通过分组来区分有不同安全特性的线程，对不同组的线程进行不同的处理，还可以通过线程组的分层结构来支持不对等安全措施的采用。Java 的 ThreadGroup 类提供了大量的方法供程序对线程组树中的每一个线程组以及线程组中的每一个线程进行操作。ThreadGroup 包含的主要方法有：

activeCount()，返回该线程组当前活动线程的估计数。

activeGroupCount()，返回该线程组内当前活动线程组的估计数。

getMaxPriority()，返回该线程组的最高优先级。

getName()，返回该线程组的名称。

getParent()，返回该线程组的双亲。

isDaemon()，测试该线程组是否是 daemon 线程组。

enumerate(Thread list[], boolean recurse)，把该线程组中所有活动线程拷贝到指定数组中。如果 recurse 标志为 true，将包含所有活动子组及其活动子组的活动子组，依次递归。如果数组太小装不下所有的线程，则忽略多出的线程。

【程序实现】

ThreadList.java 文件内容如下：

```
package thread;
import java.io.*;
import java.awt.*;
import javax.swing.*;

public class ThreadLister {
    //显示一个线程的信息
    private static void printThreadInfo(
        PrintWriter out,
        Thread t,
        String indent) {
        if (t == null)
            return;
        out.println(
            indent
                + "Thread: "
                + t.getName()
                + "  Priority: "
                + t.getPriority()
                + (t.isDaemon() ? " Daemon" : "")
                + (t.isAlive() ? "" : " Not Alive"));
    }

    //显示线程组的信息和它里面的线程或线程组的信息
    private static void printGroupInfo(
        PrintWriter out,
        ThreadGroup g,
        String indent) {
        if (g == null)
            return;
        int num_threads = g.activeCount();
        int num_groups = g.activeGroupCount();
        Thread[] threads = new Thread[num_threads];
        ThreadGroup[] groups = new ThreadGroup[num_groups];
        //将线程组的所有活动线程拷贝到指定数组，不包含活动子组
        g.enumerate(threads, false);
        g.enumerate(groups, false);
        out.println(
            indent
                + "Thread Group: "
                + g.getName()
                + "  Max Priority: "
                + g.getMaxPriority()
                + (g.isDaemon() ? " Daemon" : ""));
        for (int i = 0; i < num_threads; i++)
            printThreadInfo(out, threads[i], indent + "    ");
        for (int i = 0; i < num_groups; i++)
            printGroupInfo(out, groups[i], indent + "    ");
    }

    //寻找根线程组并把它的信息都显示出来
    public static void listAllThreads(PrintWriter out) {
        ThreadGroup current_thread_group;
        ThreadGroup root_thread_group;
        ThreadGroup parent;

        //获得当前线程组
```

```
        current_thread_group = Thread.currentThread().getThreadGroup();

        //寻找根线程组
        root_thread_group = current_thread_group;
        parent = root_thread_group.getParent();
        while (parent != null) {
            root_thread_group = parent;
            parent = parent.getParent();
        }

        //显示它的列表信息
        printGroupInfo(out, root_thread_group, "");
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("ThreadLister Demo");
        JTextArea textarea = new JTextArea();
        frame.getContentPane().add(
            new JScrollPane(textarea),
            BorderLayout.CENTER);
        frame.setSize(500, 400);
        frame.setVisible(true);

        //用 StringWriter 流获取线程列表字符串信息
        StringWriter sout = new StringWriter();        //用于获取列表
        PrintWriter out = new PrintWriter(sout);
        ThreadLister.listAllThreads(out);              //将列表置于流中
        out.close();
        String threadListing = sout.toString();        //取得线程列表信息字符串

        //将线程列表信息在窗口上显示出来
        textarea.setText(threadListing);
    }
}
```

【运行结果】

运行结果如图 2-3 所示。

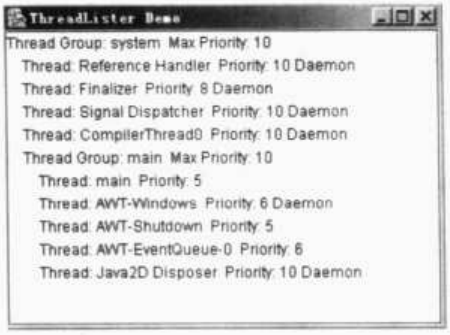


图 2-3

【分析与总结】

ThreadLister 的 listAllThreads 方法将该程序运行的线程调用情况都在面板当中显示出来。由于线程组是一棵树，所以可以用递归的方法显示出线程组树的所有线程的信息。

实例 30　生产者－消费者问题

本实例演示如何用 Java 2 解决著名的生产者 – 消费者问题。

【编程要点】

计算机系统中，每个进程都申请使用和释放各种不同类型的资源，这些资源既可以像

外设、内存及缓冲区等硬件资源，也包括临界区、数据、例程等软件资源，把系统中使用某一类资源的进程称为消费者，而把释放同类资源的进程称为该资源的生产者。

生产者 - 消费者问题是一个同步问题。把一个长度为 n 的有界缓冲区（n>0）与一群生产者进程 P1，P2，…，P3 和一群消费者进程 C1，C2，…，Ck 联系起来。生产者和消费者之间满足如下条件：

（1）消费者想接收数据时，有界缓冲区中至少有一个单元是满的。

（2）生产者想发送数据时，有界缓冲区中至少有一个单元是空的。

另外，由于有界缓冲区是临界资源，因此，各生产者进程和各消费者进程之间必须互斥执行。

【程序实现】

生产者类 Producer.java 文件内容如下：

```
package thread;
//生产者线程
public class Producer extends Thread {
    private Buffer buffer;
    private int number;

    public Producer(Buffer buffer, int number) {
        this.buffer = buffer;
        this.number = number;
    }

    public void run() {
        for (int i = 0;;) {
            buffer.put(i);
            System.out.println("生产者#" + number + "生产 " + i++);
            try {
                Thread.sleep((int) (Math.random() * 100));
            } catch (InterruptedException exc) {}
        }
    }
}
```

消费者类 Consumer.java 文件内容如下：

```
package thread;
//消费者线程
public class Consumer extends Thread {
    private Buffer buffer;
    private int number;

    public Consumer(Buffer buffer, int number) {
        this.buffer = buffer;
        this.number = number;
    }

    public void run() {
        for (;;) {
            int v = buffer.get();
            System.out.println("消费者#" + number + "消费" + v);
        }
    }
}
```

有界缓冲区类 Buffer.java 文件内容如下：

```
package thread;
//生产者与消费者共享的缓冲区，必须实现读、写的同步
public class Buffer {
    private int contents;
    private boolean available = false;

    public synchronized int get() {
```

```
        while (! available) {
            try {
                this.wait();
            } catch (InterruptedException exc) {}
        }
        int value = contents;
        //消费者取出内容，改变存取控制 available
        available = false;
        System.out.println("取出" + contents);
        this.notifyAll();
        return value;
    }

    public synchronized void put(int value) {
        while (available) {
            try {
                this.wait();
            } catch (InterruptedException exc) {}
        }
        contents = value;
        //生产者放入内容，改变存取控制 available
        available = true;
        System.out.println("放入" + contents);
        this.notifyAll();
    }
}
```

主程序 ProducerConsumerProblem.java 文件内容如下：

```
package thread;
//演示生产者-消费者问题的主程序
public class ProducerConsumerProblem {
    public static void main(String[] args) {
        Buffer buffer = new Buffer();
        new Producer(buffer, 101).start();
        new Consumer(buffer, 308).start();
        new Consumer(buffer, 309).start();
    }
}
```

【运行结果】

```
放入 0
生产者#100 生产 0
取出 0
消费者#300 消费 0
放入 1
取出 1
消费者#300 消费 1
生产者#100 生产 1
放入 2
取出 2
消费者#301 消费 2
生产者#100 生产 2
放入 3
取出 3
消费者#300 消费 3
生产者#100 生产 3
放入 4
取出 4
消费者#301 消费 4
生产者#100 生产 4
放入 5
生产者#100 生产 5
取出 5
```

```
消费者#300 消费 5
放入 6
取出 6
消费者#300 消费 6
生产者#100 生产 6
```

【分析与总结】

由程序中可以看出生产者 - 消费者问题的读写同步是在 Buffer 类中实现的，Buffer 类中通过存取控制变量 available 和两个原子操作 put 和 get 方法来实现，简单易懂。

实例 31 死锁

本实例演示多线程中由于竞争资源的而产生的死锁情况。

【编程要点】

在操作系统中，临界资源是同一时间只准一个进程使用的资源，临界区是进程访问临界资源期间所执行的那段程序。产生死锁的四个必要条件是：

（1）临界资源使用的互斥性。

（2）部分地分配资源。

（3）资源的循环等待。

（4）资源的不可抢占性。

对于 Java 而言，死锁主要是线程死锁。临界资源就是同一时间只准一个线程使用的资源，临界区是线程访问临界资源期间所执行的那段程序。Java 的 synchronized 就是用来保持临界区同步的。

【程序实现】

```
//Deadlock.java
package thread;

public class Deadlock {
    public static void main(String[] args) {
        //定义线程竞争的两个临界资源
        final Object resource1 = "resource1";
        final Object resource2 = "resource2";
        //第一个线程，先竞争 resource1，后竞争 resource2
        Thread t1 = new Thread() {
        public void run() {
            //锁定 resource1
            synchronized(resource1) {
            System.out.println("Thread 1: locked resource 1");
            //线程休眠一段时间
            try { Thread.sleep(50); }
            catch (InterruptedException e) {}

            //竞争 resource2
            synchronized(resource2) {
                System.out.println("Thread 1: locked resource 2");
            }
            }
        }
        };

        //第二个线程，先竞争 resource2，后竞争 resource1
        Thread t2 = new Thread() {
        public void run() {
            //线程锁定 resource1
            synchronized(resource2) {
```

```
                System.out.println("Thread 2: locked resource 2");

                //休眠片刻
                try { Thread.sleep(50); }
                catch (InterruptedException e) {}

                //竞争 resource2
                synchronized(resource1) {
                    System.out.println("Thread 2: locked resource 1");
                }
                }
            }
            };

        //启动两个线程
            t1.start();
            t2.start();
    }
}
```

【运行结果】

控制台输出如下内容：

```
Thread 1: locked resource 1
Thread 2: locked resource 2
```

如此可见 Thread1 和 Thread2 始终都得不到它们想要的资源，Thread1 想要的 resource2 给 Thread2 占用，Thread2 想要的 resource1 给 Thread1 占用了。

【分析与总结】

关于如何避免死锁，后面还会继续介绍一些例子。

实例 32 哲学家就餐问题

本实例解决著名的哲学家就餐问题。

【编程要点】

哲学家就餐问题简述如下：有五个哲学家围坐在一圆桌旁，桌中央有一盘通心粉，每人面前有一只空盘子，每两人之间放一只筷子，每个哲学家的行为是思考，感到饥饿，然后吃通心粉。为了吃通心粉，每个哲学家必须拿到两只筷子，并且每个人只能直接从自己的左边或右边去取筷子。当五个哲学家同时拿起右边的筷子时，就发现自己左手边的筷子都被另外一个哲学家拿走，而每个哲学家都在等待着自己左手边的筷子，于是就出现了死锁。防止死锁的方法有很多，归根到底还是防止循环等待的产生。

本实例用到的 Thread 类中的几个关键方法如下：

public static native Thread currentThread()：返回指向当前执行线程对象的指针。

Public final void setPriority(int newPriority)：改变线程优先级。首先，该线程的 checkAccess 方法被调用且无参数。这会导致抛出 SecurityException。否则，该线程的优先级被设置为指定的 newPriority 和线程组中线程所允许的最大优先级中的较小值。若优先级不在范围 MIN_PRIORITY 到 MAX_PRIORITY 中，则抛出 IllegalArgumentException。若当前线程不能修改该线程，则抛出 SecurityException。

Public final int getPriority()：返回该线程的优先级。

【程序实现】

Dinner.java 文件内容如下：

```
package thread;
```

```
class Lock {
    protected  oon an locked;                    //用来进行进程之间的同步

    public Lock() {
        locked = false;
    }

    //同步方法,每个进程取筷子之前要 locked=false 才行
    public synchronized void lock() throws InterruptedException {
        while (locked)
            wait();                              //等待 locked=false
        locked = true;                           //加锁,防止别的进程取同一支筷子
    }

    public synchronized void unlock() {          //解锁
        locked = false;
        notify();                                //唤醒别的进程
    }
}

class Fork {                                     //Fork 表示就餐用的筷子
    public char id;                              //每支筷子的标识符

    private Lock lock = new Lock();              //lock 用于同步

    public void pickup() throws InterruptedException {
        lock.lock();                             //取筷子后加锁
    }

    public void putdown() throws InterruptedException {
        lock.unlock();                           //放筷子后解锁
    }

    public Fork(int j) {                         //构造函数传进每支筷子的标识符
        Integer i = new Integer(j);
        id = i.toString().charAt(0);
    }
}

//Philosopher 代表哲学家,是 Thread 的子类,可以单独运行一个线程
class Philosopher extends Thread {
    public char state = 't';
    //state 代表哲学家的状态,初始化为思考状态

    private Fork L, R;                           //L,R 分别表示左右筷子

    //构造函数,传进当前哲学家用的左右筷子编号
    public Philosopher(Fork left, Fork right) {
        super();
        L = left;
        R = right;
    }

    protected void think() throws InterruptedException {
        sleep((long) (Math.random() * 7.0));
    }
    protected void eat() throws InterruptedException {
        sleep((long) (Math.random() * 7.0));
    }

    //run 函数开始一个线程,代表当前哲学家的活动
    public void run() {
        int i;
        try {
            for (i = 0; i < 100; i++) {          //每个哲学家循环一百次后结束
```

```
                state = 't';
                think();                    //思考
                state = 'w';
                sleep(1);
                L.pickup();                 //取左筷子
                state = 'w';
                sleep(1);
                R.pickup();                 //取右筷子
                state = 'e';
                eat();                      //就餐
                L.putdown();                //放下左筷子
                R.putdown();                //放下右筷子
            }
            state = 'd';                    //当前线程结束,当前哲学家停止一切活动
        } catch (InterruptedException e) {
        }
    }
}

public class Dinner {
    static Fork[] fork = new Fork[5];                       //五支筷子
    static Philosopher[] philo = new Philosopher[5];        //五个哲学家

    public static void main(String[] args) {
        int i, j = 0, k = 0;
         oor an  oon;

        for (i = 0; i < 5; i++) {                           //初始化每支筷子
            fork[i] = new Fork(i);
        }
        for (i = 0; i < 4; i++) {                           //初始化每个哲学家
            philo[i] = new Philosopher(fork[i], fork[(i + 1) % 5]);
        }

        //最后一个哲学家是用第一支和最后一支筷子
        philo[4] = new Philosopher(fork[0], fork[4]);

        for (i = 0; i < 5; i++) {                           //开始每个线程(哲学家开始活动)
            Philo[i].start();
        }

        int newPrio = Thread.currentThread().getPriority() + 1;
        Thread.currentThread().setPriority(newPrio);
         oon = true;
        while ( oon) {
            for (i = 0; i < 5; i++) {
                System.out.print(philo[i].state); //打印每个线程(哲学家)的状态
            }
            if (++j % 5 == 0)
                System.out.println();
            else
                System.out.print(' ');

             oon = false;
            for (i = 0; i < 5; i++) {       //如果所有线程都结束,就停止循环
             oon |= philo[i].state != 'd'; //diner[i].state = 'd'表示线程 i 已结束
            }
            try {
                Thread.sleep(51);           //让当前线程休眠
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}
```

【运行结果】

```
ttttt twewe tewew tteww wettt
wewet ettww tewwt wewet wwete
wetet wwewe wwett wetww twwew
wweww wtete wwete wewww wette
wwwwe wwede wetdw tedde weddt
tdddw ddddd
```

其中 t 表示对应的哲学家在思考，w 表示在等待，e 表示在吃东西，d 表示该线程结束。从结果可以看出每个哲学家都有吃的机会，并且不会发生死锁。

【分析与总结】

Fork 类实现当筷子被某个哲学家取走时，其他哲学家就不能再取，Philosopher 类实现哲学家的就餐线程。其防止死锁的方法是在 main 方法中的下面两个语句实现的：

```
int newPrio = Thread.currentThread().getPriority() + 1;
Thread.currentThread().setPriority(newPrio);
```

将当前线程的优先级提高，这样其他线程就不会跟当前哲学家就餐线程竞争筷子了，当前线程完成一次就餐放下筷子后又唤醒其他的哲学家就餐线程，如此继续下去。

在多线程编程中要预防死锁的发生，不然等到死锁发生后才处理就会浪费大量的时间和资源。

小结

多线程是指同时存在几个执行体，按几条不同的执行路线共同工作的情况。Java 语言内置对多线程的支持。本章重点是要掌握 Java 编程实现多线程应用的途径：一种是创建用户自己的线程子类，一种是在用户自己的类中实现 Runnable 接口。无论哪种方法，都需要用到 Java 基础类库中的 Thread 类及其方法。

第 3 章　Java 输入/输出编程

本章主要介绍 Java 的 I/O 编程，包括文件的读写、压缩以及流、管道等操作，目录列示器、目录的检查与操作、对象次第读写等编程知识。

实例 33　文件的读取

本例实现文件的读取。

【编程要点】

FileInputStream（文件输入流）是一个从 File 或 FileDescriptor 读取数据的输入流。其构造函数有：

FileInputStream(File)：创建一个输入文件流，从指定的 File 对象读取数据。

FileInputStream(FileDescriptor)：创建一个输入文件流，从指定的文件描述器中读取数据。

FileInputStream(String)：创建一个输入文件流，从指定名称的文件读取数据。

读入的方法有：

read()：从当前输入流中读取一字节数据。

read(byte[])：将当前输入流中 b.length 个字节数据读到一个字节数组中。

read(byte[], int, int)：将输入流中 len 个字节数据读入一个字节数组中。

【实例程序】

```
//ReadFile.java
package javaio;
import java.io.*;
public class ReadFile
{
    public static void main(String[] args)
    {
        try
        {
            String fileName = "D:\\test.txt"; //文件名
            FileInputStream in = new FileInputStream(new File(fileName));
            //当文件没有结束时每次读取一个字节显示
            while (in.available()>0)
            {
                System.out.print((char)in.read());
            }
            in.close();
        }
        catch (IOException e)
        {
            System.out.println(e.toString());
        }
    }
}
```

【运行结果】

本实例输出的结果是 test.txt 的文件内容，但是不能显示文件中的中文，解决方法将在其他实例中说明。如果程序抛出异常，检查文件是否存在。

【分析与总结】

读文件通常使用两个输入流，其中创建主要用于连接要读取文件的 FileInputStream 流时，可使用带绝对路径的文件名或一个文件对象与流连接；创建主要用于读取数据的 DataInputStream 流时，使用 FileInputStream 流作为输入参数即可获取数据。

实例 34 写入文件

本实例实现如何写入数据到一个文件。

【编程要点】

FileOutputStream 是向 File 或 FileDescriptor 输出数据的一个输出流。其构造函数有：

FileOutputStream(File)：创建一个文件输出流，向指定的 File 对象输出数据。

FileOutputStream(FileDescriptor)：创建一个文件输出流，向指定的文件描述器输出数据。

FileOutputStream(String)：创建一个文件输出流，向指定名称的文件输出数据。

FileOutputStream(String, boolean)：用指定系统的文件名，创建一个输出文件。

写入的方法有：

write(byte[])：将指定字节数组中 b.length 字节写入当前文件输出流。

write(byte[], int, int)：将指定字节数组中以偏移量 off 开始的 len 个字节写入当前文件输出流。

write(int)：将指定字节写入当前文件输出流。

【实例程序】

```
//WriteFile.java
package javaio;
import java.io.*;

public class WriteFile{
    public static void main(String args[]){
        try{
            String fileName = "out.txt";  //指定文件名
            File file = new File(fileName);
            FileOutputStream out = new FileOutputStream(file);  //建立输出流
            byte[] b = new byte[1024];
            String str = "Nice to see you again!";
            b = str.getBytes();  //String 转化为 byte[]
            out.write(b);  //写入文本内容
        }
        catch(IOException e){
            System.out.println(e.toString());
        }
    }
}
```

【运行结果】

本程序将在执行目录下产生一个 out.txt 的文件，文件内容为 str 字符串的内容。如果程序无法写入或者创建文件，检查当前目录是否有写权限。

【分析与总结】

写文件通常使用两个输出流，其中创建主要用于连接要写入文件的 FileOutputStream 流时，可使用带绝对路径的文件名或者一个文件对象连接与流连接；创建主要用于数据写入的 DataOutputStream 流时，使用 FileOutputStream 流作为输入参数可将数据写入指定文件。

实例 35 标准 I/O 重定向

本实例会将标准输入装置附着于一个文件，并将标准输出装置与标准事务装置重新导向至另一文件。

【编程要点】

Java System class 允许调用下列数个 static 函数，对标准输出、标准事务等 I/O streams 进行重定向：

```
setIn(InputStream)
setout(PrintStream)
setErr(PrintStream)
```

【实例程序】

```
//: Redirecting.java
import java.io.*;

public class Redirecting {
  public static void main(String[] args)
  throws IOException {
    PrintStream console = System.out;
    BufferedInputStream in = new BufferedInputStream(
      new FileInputStream("Redirecting.java"));
    PrintStream out = new PrintStream(
      new BufferedOutputStream(
        new FileOutputStream("test.out")));
    System.setIn(in);    //重定向标准输入
    System.setOut(out);  //重定向标准输出
    System.setErr(out);  //重定向标准错误
    BufferedReader br = new BufferedReader(
      new InputStreamReader(System.in));
    String s;
    while((s = br.readLine()) != null)
      System.out.println(s);
    out.close(); // 关闭输出，释放资源
    System.setOut(console);
  }
}
```

【运行结果】

在当前目录下，新产生了 test.out 文件，记录了 Redirecting.java 的内容。

【分析与总结】

如果大量输出在屏幕上造成画面快速滚动将难于阅读，或者是希望在程序中测试特定的使用者输入序列的命令行程序而言，这时，将输入内容重导向是很有用的。

实例 36 流操作

本实例演示如何创建与使用典型的 I/O 配置。

【编程要点】

I/O 程序库通常使用 stream（串流）抽象概念，所谓 stream 代表任何有能力产生出数据的数据源，或是有能力接受数据的接收端。Java 程序库之中和 I/O 相关的 classes 分为输入和输出两类。经由继承，所有衍生自 InputStream 或 Reader 的 classes，都具备了基本的 read()功能，它们可用来读取一个 byte 数据，或是读取一整个 bytes array。输出则类似。

Java1.0 版本所有和输入相关的 classes 都继承自 InputStream，所有和输出相关的 classes 都继承自 OutputStream。

InputStream 的作用是标志那些从不同起源地产生输入的类。这些起源地包括（每个都

有一个相关的 InputStream 子类）：

（1）字节数组。

（2）String 对象。

（3）文件。

（4）“管道”，它的工作原理与现实生活中的管道类似：将一些东西置入一端，它们从另一端出来。

（5）一系列其他流，以便我们将其统一收集到单独一个流内。

（6）其他起源地，如 Internet 连接等。

InputStream 的几种类型如下：

ByteArrayInputStream：允许内存中的一个缓冲区作为 InputStream 使用，从中提取字节的缓冲区，它可作为一个数据源使用。通过将其同一个 FilterInputStream 对象连接，可提供一个有用的接口。

StringBufferInputStream：将一个 String 转换成 InputStream 一个 String（字串）。基础的实施方案实际采用一个 StringBuffer（字串缓冲），它可作为一个数据源使用。通过将其同一个 FilterInputStream 对象连接，可提供一个有用的接口。

FileInputStream：用于从文件读取信息，代表文件名的一个 String，或者一个 File 或 FileDescriptor 对象，它可作为一个数据源使用。通过将其同一个 FilterInputStream 对象连接，可提供一个有用的接口。

PipedInputStream：产生为相关的 PipedOutputStream 写的数据。它实现了“管道化”的概念 PipedOutputStream，它可作为一个数据源使用。通过将其同一个 FilterInputStream 对象连接，可提供一个有用的接口。

SequenceInputStream：将两个或更多的 InputStream 对象转换成单个 InputStream 使用两个 InputStream 对象或者一个 Enumeration，用于 InputStream 对象的一个容器，它可作为一个数据源使用。通过将其同一个 FilterInputStream 对象连接，可提供一个有用的接口。

FilterInputStream：对作为 decorator 类接口使用的类进行抽象；那个 decortor 为其 InputStream 类提供了有用的功能。

OutputStream 的各种类型和 InputSteam 一一对应。

FilterInputStream 和 FileterOutStream 是两个用来“提供 decorators 以控制 InputStream 和 OutputStream”的 classes。它们都是抽象类，衍生自基类 InputStream 和 OutputStream。

DataInputStream 可以读取各种基本型别的数据和 String 对象，如果将 DataInputStream 和 DataInputStream 搭配起来就可以将基本型别的数据自某处经由 stream 移到另一处。

FilterInputStream 的各种类型：

DataInputStream 与 DataOutputStream 联合使用，使自己能以机动方式读取一个流中的基本数据类型（int、char、long 等等）InputStream，它包含了一个完整的接口，以便读取基本数据类型。

BufferedInputStream 避免每次想要更多数据时都进行物理性的读取，它将提示：“请先在缓冲区里找 InputStream”，且没有可选的缓冲区大小，其本身并不能提供一个接口，只是发出使用缓冲区的要求。要求同一个接口对象连接到一起。

LineNumberInputStream 跟踪输入流中的行号；可调用 getLineNumber()以及 setLineNumber(int)只是添加对数据行编号的能力，所以可能需要同一个真正的接口对象连接 PushbackInputStream 有一个字节的后推缓冲区，以便后推读入的上一个字符 InputStream，它可通常由编译器在扫描器中使用，因为 Java 编译器需要它。一般不在自己的代码中使用。FilterOutputStream 的各种类型如下：

DataOutputStream 与 DataInputStream 配合使用，以便采用方便的形式将基本数据类型（int、char、long 等）写入一个数据流 OutputStream，它包含了完整接口，以便我们写入基本数据类型 PrintStream 用于产生格式化输出。DataOutputStream 控制的是数据的“存储”，而 PrintStream 控制的是“显示”OutputStream，可选一个布尔参数，指示缓冲区是否与每个新行一同刷新，它对于自己的 OutputStream 对象，应该用“final”将其封闭在内。可能经常都要用到它 BufferedOutputStream 用它避免每次发出数据的时候都要进行物理性的写入，要求它“请先在缓冲区里找”。可调用 flush()，对缓冲区进行刷新 OutputStream，可选缓冲区大小/本身并不能提供一个接口，只是发出使用缓冲区的要求。需要同一个接口对象连接到一起 Readers 和 Writers、InputStream 和 OutputStream 以字节为向导，而 Reader 和 Writer 则提供相容于 Unicode 并基于字符的 I/O。两个不同的体系可以通过 Java 提供的“桥接类”搭配应用。InputStreamReader 能够将 InputStream 转换为 Reader，OutputStreamWriter 则能够将 OutputStreamf 转换为 Writer。

它们的对应关系如表 3-1 所示。

表 3-1 对应关系表

InputStream	Reader converter: InputStreamReader
OutputStream	Writer converter: OutputStreamWriter
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream	StringReader
没有对应的类	StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter
PipedInputStream	PipedReader
PipedOutputStream	PipedWriter
FilterInputStream	FilterReader
FilterOutputStream	FilterWriter（抽象类，无任何子类）
BufferedInputStream	BufferedReader（也提供 readLine()）
BufferedOutputStream	BufferedWriter
DataInputStream	使用 DataInputStream（除非你得使用 readLine()，这种情况下改用 BufferedReader）
PrintStream	PrintWriter
LineNumberInputStream	LineNumberReader
StreamTokenizer	StreamTokenizer（改用以 Reader 为参数的构造函数）
PushBackInputStream	PushBackReader

现在读者可以理解在前面的例子当中不能显示中文的原因了，下面的实例将解决这个问题。

【实例程序】

```
//: IOStreamDemo.java
package javaio;
import java.io.*;

public class IOStreamDemo {
  public static void main(String[] args)
  throws IOException {
    //1. 按行输入
    BufferedReader in =
      new BufferedReader(
        new FileReader("IOStreamDemo.java"));
    String s, s2 = new String();
    while((s = in.readLine())!= null)
      s2 += s + "\n";
    in.close();

    //2. 按行从标准输入读入
    BufferedReader stdin =
      new BufferedReader(
        new InputStreamReader(System.in));
    System.out.print("Enter a line:");
    System.out.println(stdin.readLine());

    //3. 从内存中读入
    StringReader in2 = new StringReader(s2);
    int c;
    while((c = in2.read()) != -1)
      System.out.print((char)c);

    //4. 从内存中取得格式化输入
    try {
      DataInputStream in3 =
        new DataInputStream(
          new ByteArrayInputStream(s2.getBytes()));
      while(true)
        System.out.print((char)in3.readByte());
    } catch(EOFException e) {
      System.err.println("End of stream");
    }

    //5. 输出到文件
    try {
      BufferedReader in4 =
        new BufferedReader(
          new StringReader(s2));
      PrintWriter out1 =
        new PrintWriter(
          new BufferedWriter(
            new FileWriter("IODemo.out")));
      int lineCount = 1;
      while((s = in4.readLine()) != null )
        out1.println(lineCount++ + ": " + s);
      out1.close();
    } catch(EOFException e) {
      System.err.println("End of stream");
    }

    //6. 数据的存储和恢复
    try {
      DataOutputStream out2 = new DataOutputStream(
        new BufferedOutputStream(
          new FileOutputStream("Data.txt")));
      out2.writeDouble(3.14159);
      out2.writeUTF("That was pi");
```

```
      out2.writeDouble(1.41413);
      out2.writeUTF("Square root of 2");
      out2.close();
      DataInputStream in5 = new DataInputStream(
        new BufferedInputStream(
          new FileInputStream("Data.txt")));
      //利用 DataInputStream 来写数据
      System.out.println(in5.readDouble());
      //只有 readUTF() 能恢复 Java-UTF 字符串:
      System.out.println(in5.readUTF());
      System.out.println(in5.readDouble());
      System.out.println(in5.readUTF());
    } catch(EOFException e) {
      throw new RuntimeException(e);
    }

    //7. 随机访问文件的读与写
    RandomAccessFile rf =
      new RandomAccessFile("rtest.dat", "rw");
    for(int i = 0; i < 10; i++)
      rf.writeDouble(i*1.414);
    rf.close();

    rf =
      new RandomAccessFile("rtest.dat", "rw");
    rf.seek(5*8);
    rf.writeDouble(47.0001);
    rf.close();

    rf =
      new RandomAccessFile("rtest.dat", "r");
    for(int i = 0; i < 10; i++)
      System.out.println(
        "Value " + i + ": " +
        rf.readDouble());
    rf.close();
  }
}
```

【运行结果】

```
Enter a line:      (这里输入后下一行显示输入值)
(输入值)
(IOStreamDemo.java 的程序，可以显示中文字体)
(IOStreamDemo.java 的程序，不能显示中文字体)
End of stream
3.14159
That was pi
1.41413
Square root of 2
Value 0: 0.0
Value 1: 1.414
Value 2: 2.828
Value 3: 4.242
Value 4: 5.656
Value 5: 47.0001
Value 6: 8.484
Value 7: 9.898
Value 8: 11.312
Value 9: 12.725999999999999
```

【分析与总结】

在第一段程序中，开启一个文件用于字符读取，可以使用 FileInputReader 并以一个 String 或 File 对象指定文件名。用 FileInputReader，要使这个文件带有缓冲功能，就把产生出来的 FileInputReader 传给 BufferedReader 的构造函数。用 readLine()读至文件末尾会返回 null，可用来中断 while 循环。

第二段程序示范了如何对 System.in 进行包装以读取 console（主控台）输入。System.in 是个 DataInputStream，而 BufferedReader 需要的参数是 Reader，所以需要引入 InputStreamReader 来进行转换。

第三段程序使用 String s2（内含整个文件的内容），并通过它来产生 StringReader，接着使用 read()一次读入一个字符，并将所得结果输出到 console。

第四段程序运用 DataInputStream 读取格式化数据，它是 byte-oriented 的 I/O class，因此将 String 转换为 bytes array 然后再 ByteArrayInputStream 最后给 DataInputStream。

第五段程序说明如何将数据写入文件，首先产生一个 FileWriter 连接至文件，再用 BufferedWriter 包装，以便对输出操作实施缓冲，最后用 PrintWriter 进行格式化输出，这里解决了中文字体的显示问题。

第六段程序使用 DataOutputStream 来写数据，并以 DataInputStream 来读数据。

第七段程序展示了随机访问文件的读写，使用 RandomAccessFile 就像合并使用 DataInputStream 和 DataOutputStream 一样，seek()可以在文件中四处移动，并改变文件的实值。

实例 37 管道操作

本实例实现线程间的管道操作。

【编程要点】

写入 PipedOutputStream 的数据保存到使用一个 1024 字节固定大小的 PipedInputStream 的输入缓冲区。如果该缓冲区已满，任何企图写入 PipedOutputStream 的线程将一直被阻塞至读取 PipedInputStream 的操作完成，并且删除缓冲区中的数据为止。假设线程 t 是从 PipedInputStream 读取数据的惟一线程并企图在一次调用 PipedOutputStream 的 write()方法中写 2000 个字节的数据到对应的 PipedOutputStream 中。在 t 线程被阻塞前，最多能够写入 1024 个字节。被阻塞后，读取 PipedInputStream 的操作不会出现，因为 t 是惟一的读取进程。该进程被完全阻塞，同时所有其他试图向 PipedOutputStream 写数据的线程也将如此。

为在一次 write()调用中写入多于 1024 个字节的数据，在写入的同时必须有另外的线程从 PipedInputStream 读取数据。例如写操作线程每次调用 write()方法时写入 2000 个字节，而主线程（main）从 PipedInputSteam 的循环缓冲区读取数据空出区空间。使得写线程从上次中止处重新开始，写入 2000 个字节的剩余部分。

【实例程序】

```
//PipeDemo.java
package javaio;
import java.io.*;
public class PipeDemo
{
    static PipedInputStream pipedIS = new PipedInputStream();
    static PipedOutputStream pipedOS = new PipedOutputStream();
    public static void main(String[] args)
    {
        try
        {
            pipedIS.connect(pipedOS);
        }
        catch (IOException e)
        {
```

```
            System.err.println("连接失败");
            System.exit(1);
        }
        byte[] inArray = new byte[10];
        int bytesRead = 0;
        //启动写操作线程
        startWriterThread();
        try
        {
            bytesRead = pipedIS.read(inArray, 0, 10);
            while (bytesRead != -1)
            {
                System.out.println("已经读取" + bytesRead + "字节...");
                bytesRead = pipedIS.read(inArray, 0, 10);
            }
        }
        catch (IOException e)
        {
            System.err.println("读取输入错误.");
            System.exit(1);
        }
    }
    //创建一个独立的线程
    //执行写入 PipedOutputStream 的操作
    private static void startWriterThread()
    {
        new Thread(new Runnable()
        {
            public void run()
            {
                byte[] outArray = new byte[2000];

                while (true)
                { //无终止条件的循环
                    try
                    {
                        //在该线程阻塞之前，有最多 1024 字节的数据被写入
                        pipedOS.write(outArray, 0, 2000);
                    }
                    catch (IOException e)
                    {
                        System.err.println("写操作错误");
                        System.exit(1);
                    }
                    System.out.println("    已经发送 2000 字节...");
                }
            }
        }).start();
    }
}
```

【分析与总结】

使用管道的原因是为了使每个线程始终能够保持简单，生产者线程仅将结果给一个数据流，消费者线程仅读取来自数据流的数据。使用管道可以将多个线程连接，而不必担心线程的同步问题。

实例 38 目录列示器

本实例演示如何得到一个目录下的所有文件的列表。

【编程要点】

这里利用了 DirFilter 实现了 FilenameFilter 接口，因此必须继承 accept 的方法。

【实例程序】

```
//DirList.java
package javaio;
import java.io.*;
import java.util.*;
import java.util.regex.*;

public class DirList {
  public static void main(String[] args) {
    File path = new File(".");
    String[] list;
    if(args.length == 0){
      list = path.list();
    }
    else{
      //这里将输入的参数作为过滤因子
      list = path.list(new DirFilter(args[0]));
    }
    for(int i = 0; i < list.length; i++){
      System.out.println(list[i]);
    }
  }
}

class DirFilter implements FilenameFilter {
  private Pattern pattern;
  public DirFilter(String regex) {
    pattern = Pattern.compile(regex);
  }
  public boolean accept(File dir, String name) {
    //判断名为 name 的文件是不是符合过滤条件
    return pattern.matcher( new File(name).getName() ).matches();
  }
}
```

【运行结果】

运行本实例时如果没有参数，则列出当前目录下所有文件列表，否则显示过滤后的文件列表。例如 java javaio.DirList .java，列出当前目录下的所有 .java 文件。

【分析与总结】

接口 FilenameFilter 中只有一个 accept 函数，因此所有此类对象都必须提供一个 accept()。撰写此类 class 的惟一理由，便是将 accept()提供给 list()有能力回头调用 accept()，借此决定某个文件名是否应该包括在列表之内。这个技巧通常被称作 callback（回调）。另外在 accept()函数中还运用了模式 Pattern 的类方法 matches 来判断该文件是否符合过滤条件，其中 File 类的 getName()方法保证了所处理的元素仅有文件名而不包括路径，因为它除去了所有路径信息。

实例 39 目录的检查与修改

本例演示如何创建，删除，修改目录和显示目录详细信息。

【编程要点】

File class 不仅能够用来表示所有的文件或者目录，也可以用来建立新的目录或整个目录路径（如果路径不存在的话）。也可以检视文件特性（大小，最后修改日期，读取日期，写入日期），查看 File 对象代表的是文件或目录。也可以删除文件。

【实例程序】

```
//MakeDirectories.java
package javaio;
import java.io.*;
```

```
public class MakeDirectories {
  private static void usage() {
    System.err.println(
      "Usage:MakeDirectories path1 ...\n" +
      "Creates each path\n" +
      "Usage:MakeDirectories -d path1 ...\n" +
      "Deletes each path\n" +
      "Usage:MakeDirectories -r path1 path2\n" +
      "Renames from path1 to path2");
    System.exit(1);
  }

  //显示文件或者目录的详细信息
  private static void fileData(File f) {
    System.out.println(
      "Absolute path: " + f.getAbsolutePath() +
      "\n Can read: " + f.canRead() +
      "\n Can write: " + f.canWrite() +
      "\n getName: " + f.getName() +
      "\n getParent: " + f.getParent() +
      "\n getPath: " + f.getPath() +
      "\n length: " + f.length() +
      "\n lastModified: " + f.lastModified());
    if(f.isFile())
      System.out.println("It's a file");
    else if(f.isDirectory())
      System.out.println("It's a directory");
  }

  public static void main(String[] args) {
    if(args.length < 1) usage();
    if(args[0].equals("-r")) {
      if(args.length != 3) usage();
      File
        old = new File(args[1]),
        rname = new File(args[2]);
      old.renameTo(rname);
      fileData(old);
      fileData(rname);
      return;
    }
    int count = 0;
    boolean del = false;
    if(args[0].equals("-d")) {
      count++;
      del = true;
    }
    count--;
    while(++count < args.length) {
      File f = new File(args[count]);
      if(f.exists()) {
        System.out.println(f + " exists");
        if(del) {
          System.out.println("deleting..." + f);
          f.delete();
        }
      }
      else { //目录不存在则需要建立
        if(!del) {
          f.mkdirs();
          System.out.println("created " + f);
        }
      }
      fileData(f);
    }
  }
}
```

【运行结果】

如果在命令行中输入 java MakeDirectories D:\test，且原来 D 盘不存在 test 目录，则输

出结果为：

```
created D:\test
Absolute path: D:\test
Can read: true
Can write: true
getName: test
getParent: D:\
getPath: D:\test
length: 0
lastModified: 1092303778000
It's a directory
```

【分析与总结】

实际使用此程序时会发现，想产生多复杂的目录路径都可以，因为 mkdirs()会自动去完成。

实例 40 压缩文件

【编程要点】

Java I/O 程序库提供了对 streams 进行压缩的 class，这些 class 并非衍生于 Reader 与 Writer，而是位于 InputStream 和 OutputStream 的继承体系内，这是因为压缩用的程序库所处理的仍是 bytes 而非字符。

【实例程序】

```
//ZipCompress.java

import java.io.*;
import java.util.*;
import java.util.zip.*;

public class ZipCompress {
  //抛出异常到控制台：
  public static void main(String[] args)
  throws IOException {
    FileOutputStream f =
      new FileOutputStream("test.zip");              //建立输出文件流
    CheckedOutputStream csum = new CheckedOutputStream(
        f, new Adler32());                           //建立冗余验证流
    ZipOutputStream out =
      new ZipOutputStream(
        new BufferedOutputStream(csum));             //建立 zip 流
    out.setComment("A test of Java Zipping");
    for(int i = 0; i < args.length; i++) {
      System.out.println(
        "Writing file " + args[i]);
      BufferedReader in =
        new BufferedReader(
          new FileReader(args[i]));
      out.putNextEntry(new ZipEntry(args[i]));       //建立压缩实体
      int c;
      while((c = in.read()) != -1){
        out.write(c);
      }
      in.close();                                    //关闭文件流，释放资源
    }
    out.close();
    System.out.println("Checksum: " +
      csum.getChecksum().getValue());                //输出效验码

    //解压缩
    System.out.println("Reading file");
    FileInputStream fi =
```

```
        new FileInputStream("test.zip");
      CheckedInputStream csumi =
        new CheckedInputStream(
          fi, new Adler32());
      ZipInputStream in2 =
        new ZipInputStream(
          new BufferedInputStream(csumi));
      ZipEntry ze;
      while((ze = in2.getNextEntry()) != null) {
        System.out.println("Reading file " + ze);
        int x;
        while((x = in2.read()) != -1)
          System.out.write(x);
      }
      System.out.println("Checksum: " +
        csumi.getChecksum().getValue());
      in2.close();

      //输出解压缩后的各个文件名
      ZipFile zf = new ZipFile("test.zip");
      Enumeration e = zf.entries();
      while(e.hasMoreElements()) {
        ZipEntry ze2 = (ZipEntry)e.nextElement();
        System.out.println("File: " + ze2);
      }
    }
}
```

【运行结果】

在命令行中输入 java ZipCompress ZipCompress.java 后，该程序输出：

```
Writing file ZipCompress.java
Checksum: 797741269
Reading file
Reading file ZipCompress.java
(接下来是 ZipCompress.java 的代码，这里省去)
Checksum:797741269
File: ZipCompress.java
```

【分析与总结】

此程序可以对任意个文件进行压缩和解压缩，每一个需要添加压缩包的文件都必须调用 putNextEntry()方法并将其传递给一个 ZipEntry 对象。该对象包含一个功能全面的接口，用其可获取和设置 Zip 文件中特定入口上可接受的所有数据，即文件名、压缩后和压缩前的长度、日期、CRC 效验和额外字段的数据、注释、压缩方法，以及是否有一个目录接口。

实例 41 对象次第读写

本实例演示了如何将对象写入到文件中，然后再从文件中还原为原对象而不丢失信息。

【编程要点】

Java 的 object serialization 机制允许程序员可以将任何实现出 Serializable 接口的对象转换为 bytes 数据。这些数据在日后又可以还原为原先的对象状态。

【实例程序】

```
//: :Worm.java
import java.io.*;

class Data implements Serializable {
  private int i;
  Data(int x) { i = x; }
  public String toString() {
    return Integer.toString(i);
  }
```

```
}

public class Worm implements Serializable {
  //产生随机正整数(小于 10)
  private static int r() {
    return (int)(Math.random() * 10);
  }
  private Data[] d = {
    new Data(r()), new Data(r()), new Data(r())
  };
  private Worm next;
  private char c;
  //i 表示 worm 链中的第几段
  Worm(int i, char x) {
    System.out.println(" Worm constructor: " + i);
    c = x;
    if(--i > 0)
      next = new Worm(i, (char)(x + 1));
  }
  Worm() {
    System.out.println("Default constructor");
  }
  public String toString() {
    String s = ":" + c + "(";
    for(int i = 0; i < d.length; i++)
      s += d[i].toString();
    s += ")";
    if(next != null)
      s += next.toString();
    return s;
  }
  //抛出异常到控制台
  public static void main(String[] args)
  throws ClassNotFoundException, IOException {
    Worm w = new Worm(6, 'a');
    System.out.println("w = " + w);
    ObjectOutputStream out =
      new ObjectOutputStream(
        new FileOutputStream("worm.out"));
    out.writeObject("Worm storage");
    out.writeObject(w);  //将产生的对象写入到 worm.out 文件
    out.close();
    ObjectInputStream in =
      new ObjectInputStream(
        new FileInputStream("worm.out"));
    String s = (String)in.readObject();  //从 worm.out 中读取对象
    Worm w2 = (Worm)in.readObject();
    System.out.println(s + ", w2 = " + w2);
    //另一种用 ByteArray 作为读写对象的次第读写方法
    ByteArrayOutputStream bout =
      new ByteArrayOutputStream();
    ObjectOutputStream out2 =
      new ObjectOutputStream(bout);
    out2.writeObject("Worm storage");
    out2.writeObject(w);
    out2.flush();
    ObjectInputStream in2 =
      new ObjectInputStream(
        new ByteArrayInputStream(
          bout.toByteArray()));
    s = (String)in2.readObject();
    Worm w3 = (Worm)in2.readObject();
    System.out.println(s + ", w3 = " + w3);
  }
}
```

【运行结果】

```
Worm constructor: 6
Worm constructor: 5
```

```
Worm constructor: 4
Worm constructor: 3
Worm constructor: 2
Worm constructor: 1
w = :a(801):b(551):c(537):d(421):e(671):f(913)
Worm storage, w2 = :a(801):b(551):c(537):d(421):e(671):f(913)
Worm storage, w3 = :a(801):b(551):c(537):d(421):e(671):f(913)
```

【分析与总结】

如果想对某个对象执行 serialize 动作，就得产生某种 OutputStream 对象。并以 ObjectOutputStream 对象加以包装。然后只要调用 writeObject()就可达到目的。输出数据将被送至该 OutputStream 中。Object serialization 不仅存储对象在内存中的原始数据，还会追踪该对象内含的所有 references 并存储它们所指的对象，而后者再追踪后者（对象）内各自包含的 suoyouireferences，反复进行同样动作，依次类推。该程序就是先制造一条将对象串接起来的“蠕虫”，每个对象都具有向下一部分的链接，以及一个由 reference 形成的 array，其中每个 references 指向一个 Data 对象。然后演示了进行次第读写后这些信息并未失踪。

小结

Java 的输入/输出功能必须借助输入/输出类库 java.io 包来实现，这个包的中的类大部分是用来完成流式输入/输出的流类。本章重点掌握各种流操作。

流是指在计算机的输入与输出之间运动的数据的序列，输入流代表从外设流入计算机的数据序列；输出流代表从计算机流向外设的数据序列。流式输入/输出方式，它最大的特点是数据的获取和发送均沿数据序列顺序进行。通过本章有助于掌握基本输入/输出流：InputStream、OutputStream 和几种具体的输入/输出流 FilterInputStream、FilterOutputStream、DataInputStream、DataOutputStream、FileInputStream、FileOutputStream、PipedInputStream、PipedOutputStream 等。

第 4 章　Java Applet 编程

Java 程序可以分为 Java Application 和 Java Applet 两类，Java Application 是由独立的解释器程序运行的，Java Applet 是另一类非常重要的 Java 程序，它不是一类可以独立运行的程序，它的字节码文件必须嵌入到另一种语言 HTML 的文件中并由负责解释 HTML 文件的浏览器充当解释器，来解释执行 Java Applet 的字节码程序。

本章通过 7 个实例介绍了 Applet 编程的基础，其中包括简单的 Applet 程序、文本域、多行文本域和它们的触发事件以及在 Applet 中的应用多线程实现动画的编程。

实例 42　简单的 Applet 程序

本实例演示 Applet 程序设计的基本内容。

【编程要点】

Java 2 中，通过所谓的 Java Foundation Classes（JFC，其中 GUI 相关部分称为 Swing）从根本上取代了 Java1.0 中的 Abstract Windows Toolkit（抽象窗口工具组，AWT）。在现代用户界面所见到的所有组件中，从图形按钮到树状显示和表格，Swing 一应俱全。它是一个庞大的程序库，它的设计理念是带给你适当的复杂度——如果某件事情够简单，你不需要撰写太多程序代码；但是如果企图进行更复杂的动作，程序代码复杂度就会成比例地增加。这意味着它的门槛虽低，却又能够提供程序员更高的要求。基于速度考虑，Swing 的所有组件都是轻量级的（lightweight），也即它不含任何特定平台代码，但它包括特定操作系统的图形控件（AWT 之所以是重量级的，是因为它含特定操作系统的代码）。为了可携性，Swing 完全以 Java 写成。

Java 的设计目标之一便是开发 applets。所谓 applets 是一种执行于 Web 浏览器上的小程序。由于 applets 对系统而言绝对安全，所以能够做的事情也是有限的。即使如此，对客户端编程（开发 Web 应用程序时的主要议题）而言，applets 仍然是极具威力的工具。

【实例程序】

```
import java.awt.*;
import javax.swing.*;
public class SimpleApplet extends JApplet{
    private String text = "这是一个简单的小应用程序.....";
    private int count = 0;
    public void init() {
        System.out.println(count + ": init() ..."); count++;
        setBackground(Color.cyan);
    }
    public void start() {
        System.out.println(count + ": start() ..."); count++;
    }
    public void stop() {
        System.out.println(count + ": stop() ..."); count++;
    }
    public void destroy() {
        System.out.println(count + ": destroy() ..."); count++;
    }
    public void paint(Graphics g) {
        System.out.println(count + ": painting() ..."); count++;
        g.setColor(Color.blue);
        g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);
```

```
            g.setColor(Color.red);
            g.drawString(text, 15, 25);
        }
}
```

【运行结果】

运行结果如图 4-1 所示。



图 4-1

控制台输出：

```
0: init() ...
1: start() ...
2: painting() ...
3: stop() ...
4: destroy() ...
```

【分析与总结】

Java 标准程序库中的一种应用框架（application framework）提供了某个特定类型下所有应用程序都需要的基本行为模式。只需要继承 application class，覆写必要的函数，即可制订自己的程序行为。Applets 便是通过应用程序框架开发而得的，所以得继承 class JApplet，并覆写必要的函数：

init()：程序片第一次被创建，初次运行初始化程序片时调用。

start()：每当程序片进入 Web 浏览器中，并且允许程序片启动它的常规操作时调用（特殊的程序片被 stop()关闭）；同样在 init()后调用。

paint()：基础类 Component 的一部分（继承结构中上溯三级）。作为 update()的一部分调用，以便对程序片的画布进行特殊的描绘。

stop()：每次程序片从 Web 浏览器的视线中离开时调用，使程序片能关闭代价高昂的操作；同样在调用 destroy()前调用。

destroy()：程序片不再需要，将它从页面中卸载时调用，以执行资源的最后清除工作。

现在来看一看 paint()方法。一旦 Component 决定自己需要更新，就会调用这个方法——可能是由于它再次回转屏幕，首次在屏幕上显示，或者是由于其他窗口临时覆盖了 Web 浏览器。此时程序片会调用它的 update()方法（在基础类 Component 中定义），该方法会恢复一切该恢复的东西，而调用 paint()正是这个过程的一部分。没必要对 paint()进行过载处理，但构建一个简单的程序片无疑是方便的方法，所以我们首先从 paint()方法开始。

update()调用 paint()时，会向其传递指向 Graphics 对象的一个句柄，那个对象代表准备在上面描绘（作图）的表面。这是非常重要的，因为受到项目组件的外观的限制，因此不能画到区域外，这可是一件好事，否则就会画到线外去。在程序片的例子中，程序片的外观就是这界定的区域。

同样，可对一系列的图行对象进行操作。这些操作都与在画布上作图有关。所以其中

的大部分都要涉及图像、几何形状、圆弧等等的描绘（注意如果读者有兴趣，可在 Java 文档中找到更详细的说明）。利用有些方法可画出字符，而其中最常用的就是 drawString()。对于它，需指出自己想描绘的 String（字串），并指定它在程序片作图区域的起点。这个位置用像素表示，所以它在不同的机器上看起来是不同的，但至少是可以移植的。其中 drawRect()函数是画矩形的函数，四个参数分别表示左上角顶点 x 坐标、y 坐标，矩形宽度、高度。setColor()函数是设定画布的颜色，这样就可以指定画出的矩形或显示的字符串的颜色。

实例 43　文本域

本实例演示文本域的简单使用。

【编程要点】

TextField 对象表示可编辑的单行文本的文本组件。其构造函数：

TextField()：创建一个新的文本域。

TextField(int)：用指定列数创建一个新的空文本域。

TextField(String)：创建一个新的文本域，并用指定文本初始化。

TextField(String, int)：创建一个新的文本域，并用待显示的指定文本初始化，且它的宽度足以显示指定字符数。

本例用到的方法有：

setEchoChar(char)：设置当前文本域的回响字符，一般用于密码框。

setEditable(boolean)：设置当前文本域能否编辑。

【程序实现】

TextFieldDemo.java 文件内容如下：

```
import java.applet.*;
import java.awt.*;

public class TextFieldDemo extends Applet {
    TextField text1, text2, text3;

    public void init() {
        text1 = new TextField("输入密码：", 10);
        text1.setEditable(false);
        text2 = new TextField(10);
        text2.setEchoChar('*');
        text3 = new TextField("您好", 20);
        add(text1);
        add(text2);
        add(text3);
        text3.setText("");
    }
}
```

【运行结果】

运行结果如图 4-2 所示。

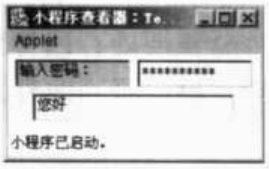


图 4-2

【分析与总结】

文本域是比较常用的组件之一，读者应该掌握它的使用方法。

实例 44 文本域触发事件

本实例演示如何处理文本域事件。

【编程要点】

事件是一个携带信息的对象，标识事件的种类以及产生事件的事件源。文本域事件触发的过程如下，当用户在文本域内敲下一个键时，AWT 将向文本域发送两个动作事件。第一个表示键按下，第二个表示键释放。每个动作事件都体现了某时刻的系统状态。每个动作事件的属性将指示哪一个键被按下、哪个修饰键也被按下和此事件发生的时刻。

因为此事件是 ActionEvent 的一个实例，所以类 ActionEvent 的 processEvent 方法将检查该事件，并将它传递给 processActionEvent 方法。后面的方法又将该事件重定向到任一个对此文本域产生的动作事件感兴趣的 ActionListener 对象。

【程序实现】

TextFieldEvent.java 文件内容如下：

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class TextFieldEvent extends Applet implements ActionListener {
    TextField text1, text2, text3;

    public void init() {
        text1 = new TextField(10);
        text2 = new TextField(10);
        text3 = new TextField(20);
        add(text1);
        add(text2);
        add(text3);
        text1.addActionListener(this); //将主类的实例作为 text1 的监视器，因此主类必须实现接口 ActionListener
        text2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == text1) {
            String word = text1.getText();
            if (word.equals("boy")) {
                text3.setText("男孩");
            } else if (word.equals("girl")) {
                text3.setText("女孩");
            } else if (word.equals("sun")) {
                text3.setText("太阳");
            } else {
                text3.setText("没有该单词");
            }
        } else if (e.getSource() == text2) {
            String word = text2.getText();
            if (word.equals("男孩")) {
                text3.setText("boy");
            } else if (word.equals("女孩")) {
                text3.setText("girl");
            } else if (word.equals("太阳")) {
                text3.setText("sun");
            } else {
```

```
                    text3.setText("没有该单词");
                }
            }
        }
    }
```

【运行结果】

运行结果如图 4-3 所示。

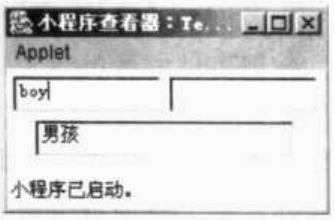


图 4-3

当在左上方的文本域中输入英文 boy 时，下方的文本域出现对应的中文意思，当在右上方的文本域中输入中文时，在下方的文本域出现对应的英文单词。这里只实现了三个单词和中文的对应关系，其他找不到的单词显示“没有该单词”。

【分析与总结】

TextField 的事件是比较常要处理的事件，读者应该掌握它的事件处理方法。

实例 45　多行文本域

本实例演示多行文本域的使用。

【编程要点】

TextArea 对象显示多行文本的区域。它可以被设置为可编辑的或只读的。其构造函数如下：

TextArea()：创建一个新的文本区域。

TextArea(int, int)：用指定的行、列数创建一个新的文本区域。

TextArea(String)：用指定文本，创建一个新的文本区域。

TextArea(String, int, int)：用指定的文本和行、列数，创建一个新的文本区域。

TextArea(String, int, int, int)：用指定的文本、行列数和可视的滚动条，创建一个新的文本区域。

常用的方法如下：

insert(String, int)：在当前文本区域内的指定位置，插入指定文本。

append(String)：向当前文本区域追加给定文本。

replaceRange(String, int, int)：用指定的置换文本替换指定的开始和结束位置间的文本。

【程序实现】

TextAreaDemo.java 文件内容如下：

```
import java.applet.*;
import java.awt.*;

public class TextAreaDemo extends Applet {
    TextArea text1, text2;

    public void init() {
        text1 = new TextArea("I am a student", 6, 16);
        text2 = new TextArea(6, 16);
```

```
        add(text1);
        add(text2);
        //将字符串加到文本域的末尾
        text2.append("I am learning programing.");
        //在文本域的指定位置插入字符串
        text2.insert("Java ", 14);
        //选定文本域 text1 中的所有内容
        text1.selectAll();
        int length = text2.getText().length();
        //设定文本域 text2 中指定位置的字符串
        text2.setSelectionStart(2);
        text2.setSelectionEnd(length);
    }
}
```

【运行结果】

运行结果如图 4-4 所示。

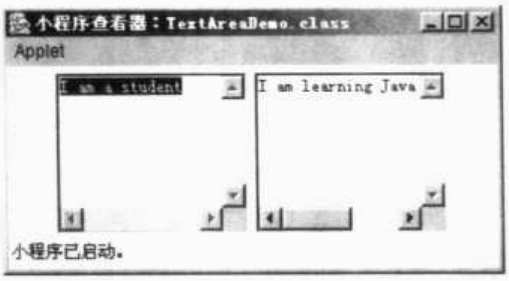


图 4-4

【分析与总结】

可以同 TextField 比较有什么异同，加深认识。

实例 46 TextArea 触发事件

本实例演示 TextArea 触发事件的处理。

【编程要点】

当 TextArea 的内容发生改变时，AWT 就会触发一个 textValueChanged 事件，用户可以用 TextListener 来侦听该事件，并捕获进行处理。

【程序实现】

TextAreaEvent.java 文件内容如下：

```
import java.util.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class TextAreaEvent extends Applet implements TextListener {
    TextArea text1, text2;

    public void init() {
        text1 = new TextArea(6, 15);
        text2 = new TextArea(6, 15);
        add(text1);
        add(text2);
        text2.setEditable(false);
        text1.addTextListener(this);
    }

    public void textValueChanged(TextEvent e) {
        if (e.getSource() == text1) {
            String s = text1.getText();
```

```
                StringTokenizer fenxi = new StringTokenizer(s, " ,'\n'");
                int n = fenxi.countTokens();
                String a[] = new String[n];
                for (int i = 0; i <= n - 1; i++) {
                    String temp = fenxi.nextToken();
                    a[i] = temp;
                }
                for (int i = 0; i <= n - 1; i++) //按字典序从小到大排序
                {
                    for (int j = i + 1; j <= n - 1; j++) {
                        if (a[j].compareTo(a[i]) < 0) {
                            String t = a[j];
                            a[j] = a[i];
                            a[i] = t;
                        }
                    }
                }
                text2.setText(null); //刷新显示
                for (int i = 0; i < n; i++) {
                    text2.append(a[i] + "\n");
                }
            }
        }
    }
```

【运行结果】

运行结果如图 4-5 所示。

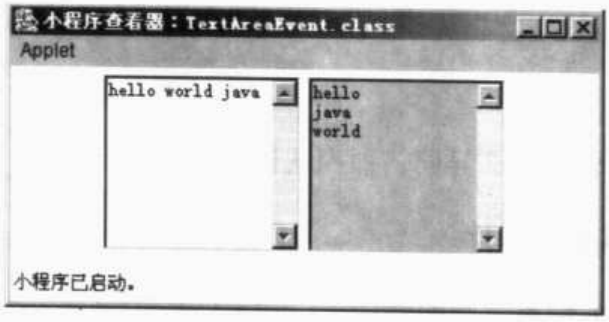


图 4-5

上面的文本域输入的字符串在下面那个文本域也会即时显现，并按字典序列排列。

【分析与总结】

程序中用到了 StringTokenizer 类，StringTokenizer 类允许一个应用把字符串分解成一个个标记。标记化方法比 StreamTokenizer 类使用的方法简单的多。StringTokenizer 方法不区分标识符、数字和引用字符串，它们也不识别并且跳过注释。分隔符集合（分割标记的字符）可在创建时间指定或在每个标记基础上指定。其构造函数：

public StringTokenizer(String str, String delim)，为指定字符串构造一个 string tokenizer 对象。在 delim 参数中的字符是分隔标记的分隔符。

实例 47　画红点

本实例演示在 Applet 中画点。

【编程要点】

Graphics 类是所有图形上下文的抽象基类，这个上下文允许应用将图形绘制到由不同设备实现的组件上，以及绘制到空闲屏幕的映像中。这里用到 Graphics 的几个方法：

public abstract void fillOval(int x, int y, int width, int height)：用当前颜色填充由指定矩

形限定的椭圆。

public void drawRect(int x, int y, int width, int height)：绘制指定矩形的轮廓。矩形的左边和右边分别为 x 和 x + width。顶部边沿和底部边沿分别为 y 和 y + height。该矩形是使用图形上下文的当前颜色绘制的。

public abstract void fillRect(int x, int y, int width, int height)：填充指定的矩形。矩形的左边和右边分别为 x 和 x + width - 1。顶部边沿和底部边沿分别为 y 和 y + height - 1。结果矩形覆盖了一个宽 width 个像素高 height 个像素的区域。使用图形上下文的当前颜色来填充矩形。

【程序实现】

Spot.java 文件内容如下：

```
public class Spot {
    public int size;
    public int x, y;

    public Spot(int intSize) {
        size = intSize;
        x = -1;
        y = -1;
    }
}
```

ClickMe.java 文件内容如下：

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class ClickMe extends Applet implements MouseListener {
    private Spot spot = null;
    private static final int RADIUS = 7;

    public void init() {
     addMouseListener(this);
    }

    public void paint(Graphics g) {
     //画一个有黑色边框，并且背景色为白色的区域
        g.setColor(Color.white);
     g.fillRect(0, 0, getSize().width - 1, getSize().height - 1);
        g.setColor(Color.black);
     g.drawRect(0, 0, getSize().width - 1, getSize().height - 1);

     //画点
        g.setColor(Color.red);
     if (spot != null) {
         g.fillOval(spot.x - RADIUS, spot.y - RADIUS, RADIUS * 2, RADIUS * 2);
        }
    }
    public void mousePressed(MouseEvent event) {
        if (spot == null) {
            spot = new Spot(RADIUS);
        }
     spot.x = event.getX();
     spot.y = event.getY();
     repaint();
    }
    public void mouseClicked(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

【运行结果】

运行结果如图 4-6 所示。



图 4-6

【分析与总结】

本实例的画点其实是通过填充一个小圆来实现的。

实例 48 渐显文字

本实例演示一个在 Applet 中文字渐显的效果。

【编程要点】

通过多线程来实现这一动画效果。这里还用到消除图像闪烁的双缓冲技术，利用此技术制作出来的动画能最大限度地避免闪烁现象，这种技术的缺点是需要占用大量的内存。

双缓冲技术的工作过程：在屏幕外面创建一个虚拟的备用屏幕，直接在备用屏幕上面作画，画完以后就将备用屏幕中的点阵内容直接切换给当前屏幕，从而提高了作画的速度。

【程序实现】

Fader.java 文件内容如下：

```
import java.awt.*;
import java.applet.Applet;
//定义 Fader 类
public class Fader extends Applet implements Runnable {
    Dimension d;                    //定义一个 Dimension 对象
    Image ii;                       //定义一个 Image 对象
    Graphics goff;                  //定义一个 Graphics 对象
    Thread thethread;               //定义一个线程

    //定义一个字体对象
    Font font = new Font("Helvetica", Font.BOLD, 36);
    FontMetrics fm;                 //定义一个字体矩阵
    String s1;
    String s2;
    int color1, color2;
    int dcolor1, dcolor2;

    //获取小应用程序信息
    public String getAppletInfo() {
        return ("Text fader");
    }

    //初始化
    public void init() {
        Graphics g;
        int i;
        d = size();                         //获取小应用程序大小
        g = getGraphics();                  //为画图做准备
        g.setFont(font);                    //设置字体
        fm = g.getFontMetrics();            //获取字体矩阵
```

```
        setBackground(Color.black);     //设置背景色
        s1 = "Welcome";                 //所要显示的文字
        s2 = "Best Wishes";             //所要显示的文字
        color1 = 10;
        color2 = 245;
        dcolor2 = -2;
        dcolor1 = 2;
    }
    //paint()方法
    public void paint(Graphics g) {
        if (goff == null && d.width > 0 && d.height > 0) {
            //创建一个可以使用的双缓冲图像
            ii = createImage(d.width, d.height);
            goff = ii.getGraphics();
        }
        if (goff == null || ii == null)
            return;
        g.setFont(font);
        goff.setFont(font);
        if (color1 < color2) {
            //设置颜色，并将文字画在屏幕上
            goff.setColor(new Color(color1 / 4, color1 / 2, color1));
            goff.drawString(s1, (d.width - fm.stringWidth(s1)) / 2,d.height / 2);
            goff.setColor(new Color(color2 / 4, color2 / 2, color2));
            goff.drawString(s2, (d.width - fm.stringWidth(s2)) / 2,d.height / 2);
        } else {
            goff.setColor(new Color(color2 / 4, color2 / 2, color2));
            goff.drawString(s2, (d.width - fm.stringWidth(s2)) / 2,d.height / 2);
            goff.setColor(new Color(color1 / 4, color1 / 2, color1));
            goff.drawString(s1, (d.width - fm.stringWidth(s1)) / 2,d.height / 2);
        }
        g.drawImage(ii, 0, 0, this);
        //变换颜色
        color1 += dcolor1;
        color2 += dcolor2;
        if (color1 <= 3 || color1 >= 250)
            dcolor1 = -dcolor1;
        if (color2 <= 3 || color2 >= 250)
            dcolor2 = -dcolor2;
    }
    //run()方法
    public void run() {
        long starttime;
        Graphics g;
        //设置线程优先权
        Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
        g = getGraphics();
        while (true) {
            starttime = System.currentTimeMillis(); //获取系统当前时间
            try {
                paint(g);
                starttime += 20;
                //使线程睡眠
                Thread.sleep(Math.max(0,                starttime
System.currentTimeMillis()));
            } catch (InterruptedException e)        //抛出异常
            {
                break;
            }
        }
    }
    //启动线程
    public void start() {
        if (thethread == null) {
            thethread = new Thread(this);
            thethread.start();
        }
```

```
    }
    //停止线程
    public void stop() {
        if (thethread != null) {
            thethread.stop();
            thethread = null;
        }
    }
}
```

【运行结果】

运行结果如图 4-7 所示。



图 4-7

【分析与总结】

本实例双缓冲技术的实现过程：

```
Image ii;          //定义一个 Image 对象
Graphics goff;     //定义一个 Graphics 对象
ii = createImage(d.width, d.height);
goff = ii.getGraphics();
```

然后在 goff 上 draw，最后调用 g.drawImage(ii, 0, 0, this)将 ii 画在屏幕上。

小结

本章重点介绍了 Applet 编程的方法，主要应该掌握 java.awt 和 java.awt.event 两个包的使用。

第 5 章　Java Swing 图形界面编程

在 Java 2 中，javax.swing 包被列入 Java 的基础类库（JFC），其中定义的 Swing GUI 组件相对于 java.awt 包的各种 GUI 组件增加了许多功能。本章通过 16 个实例介绍了 Swing 的各种组件：JFrame、JButton、JScrollPane、JCheckBox、JList、JTree、JTable 等，还有菜单、对话框等常用界面设计组件，最后通过综合的实例，计算器、TicTacToe 游戏、文本编辑器，进一步深化了 Swing 图形界面编程的认识。

实例 49　鼠标画线

本程序演示鼠标事件处理。

【编程要点】

事件是一个携带信息的对象，标识事件的种类以及产生事件的事件源。用户利用鼠标单击、拖动、按下、放开等操作时都会产生一个事件。事件需由相应的监听器接收并处理。不同的行为导致不同类型的事件发生：

ActionListener：用户单击鼠标，在文本字段区按回车键，选择一个菜单项等。

WindowListener：用户关闭一个画框（主窗口）或对话框。

MouseListener：用户单击鼠标。

MouseMotionListener：用户在一个组件上拖动鼠标。

ComponentListener：组件变为可见的。

FocusListener：组件获得键盘输入焦点。

ListSelectionListener：表格或列表当前选中项目被修改。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/*为了达到画线的功能，我们分别 implements MouseListener 与 MouseMotionListener。*/
public class MouseDemo extends JFrame implements MouseListener,
        MouseMotionListener {
    int flag;   //flag=1 代表 Mouse Moved,flag=2 代表 Mouse Dragged
    int x = 0;
    int y = 0;
    int startx, starty, endx, endy;  //起始坐标与终点坐标
    public MouseDemo() {
        Container contentPane = getContentPane();  //得到 Frame 的内容窗格
        contentPane.addMouseListener(this);
        contentPane.addMouseMotionListener(this);
        setSize(300, 300);
        show();
        addWindowListener(new WindowAdapter() {    //添加关闭 JFrame 窗口事件触发器
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
    /*由 mousePressed(),mouseReleased()取得示拖曳的开始与结束坐标*/
    public void mousePressed(MouseEvent e) {
        startx = e.getX();
```

```
            starty = e.getY();
        }

        public void mouseReleased(MouseEvent e) {
            endx = e.getX();
            endy = e.getY();
        }

        public void mouseEntered(MouseEvent e) {
        }

        public void mouseExited(MouseEvent e) {
        }

        public void mouseClicked(MouseEvent e) {
        }

        /*mouseMoved(),mouseDragged()取得鼠标移动的每一个坐标，并调用 repaint()方法*/
        public void mouseMoved(MouseEvent e) {
            flag = 1;
            x = e.getX();
            y = e.getY();
            repaint();
        }

        public void mouseDragged(MouseEvent e) {
            flag = 2;
            x = e.getX();
            y = e.getY();
            repaint();
        }

        public void update(Graphics g) {
            g.setColor(this.getBackground());
            g.fillRect(0, 0, getWidth(), getHeight());
            paint(g);
        }

        public void paint(Graphics g) {
            g.setColor(Color.black);
            if (flag == 1) {
                g.drawString("鼠标坐标：(" + x + "," + y + ")", 10, 50);
                g.drawLine(startx, starty, endx, endy);
            }
            if (flag == 2) {
                g.drawString("拖曳鼠标价坐标：(" + x + "," + y + ")", 10, 50);
                g.drawLine(startx, starty, x, y);
            }
        }

        public static void main(String[] args) {
            new MouseDemo();
        }
}
```

【运行结果】

运行结果如图 5-1 所示。

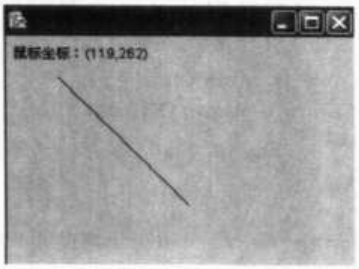


图 5-1

窗口中显示鼠标当前的坐标，当按下鼠标左键并开始拖动时，窗口中会画出一条从开始位置到结束位置的一条线段。

【分析与总结】

该程序通过事件监听器来实现事件的处理。因为跟踪鼠标移动的系统开销远大于跟踪鼠标其他行为，所以分为跟踪鼠标移动的监听器 MouseMotionListener 和跟踪其他鼠标行为的 MouseListener。其中 MouseMotionListener 包括 mouseMoved()接口和 mouseDragged()接口，其余都在 MouseListener 中。

此程序通过 mousePress()中得到鼠标的起始位置，并记录在（startX，startY）中，通过 mouseRealse()中得到鼠标的结束位置，并记录在（endX，endY）中；通过 mouseMoved()得到鼠标是否移动以及鼠标当前的位置，并执行重画操作 repaint()；通过 mouseDragged()得到鼠标是否拖动以及鼠标当前的位置，并执行重画操作 repaint()。

系统中 repaint()方法会调用 update()方法，而 update()方法中，先清空当前的画布，然后再执行 paint()操作。执行 paint()时如果鼠标是移动，则显示出鼠标的当前坐标，和从起始位置到结束位置的线段；如果是拖动，则显示出鼠标的当前坐标，和从起始位置到当前位置的线段。

最后注意的是要把监听器加到内容窗格中，即在 MouseDemo 类的构造函数中 addMouseListener，这样就大功告成了。

实例 50 使用画框

本实例演示如何使用 JFrame。

【编程要点】

Swing 组件可分为顶层容器、中间容器、原子组件。顶层容器的作用是为其他 GUI 组件的绘制提供位置，常用的顶层容器包括：JFrame、JDialog 和 JApplet。每个顶层容器都包含一个中间容器，称为内容窗格（contentpane）。

内容窗格直接或间接包含窗口中所有可见的组件（除菜单条外），被包含的 GUI（即原子组件）通过各种 add()方法添加到容器中。中间容器的目的仅仅是为了更方便地定位其他 GUI 组件，常用的中间容器包括 JPanel、JScrollPane 和 JTabbedPane。Swing 提供了大量的原子组件：JButton、JTextField、JTable、Jlabel 等等。除顶层容器外，所有的 J*组件都由 JComponent 类派生。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FrameDemo {
    public static void main(String args[]){
        JFrame frame = new JFrame("画框显示");
        //使用窗口监听器结束程序
        frame.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        JLabel label = new JLabel("JFrame");                    //创建一个文本标签
        label.setPreferredSize(new Dimension(200,100));         //设定 label 的大小
```

```
        frame.getContentPane().add(label, BorderLayout.CENTER);    //设置布局
        frame.pack();    //使 JFrame 的大小自动适合所含组件大小
        frame.setVisible(true);                                    //设定为可见的
    }
}
```

【运行结果】

运行结果如图 5-2 所示。



图 5-2

【分析与总结】

JFrame 在初始创建时不可见，要调用 setVisiable(true)来设定可见性，JFrame 的窗口标题还可以调用 setTitle(" ")来设置。另外关闭窗口的方法除了程序中的设定监听器外，还可以直接调用 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)方法来设置用户关闭窗口的缺省方法。

另外，通过布局设置确定 JLabel 组件在 JFrame 的位置。BorderLayout 使得在没有任何其他提示的条件下，无论你加入任何组件，它都一律接受，并将组件置于中央，然后伸展组件，使它与 Container 的边缘对齐；BoderLayout 还将版面分为四分边缘区和一份中央区，分别用 BorderLayout.NORTH，BorderLayout.SOUTH，BorderLayout.EAST，BorderLayout.WEST 和 BorderLayout.CENTER。读者可从 JFrame 中自行增加组件来观察这一布局。

实例 51 JButton 示例

本实例演示按钮的添加和事件的响应。

【编程要点】

利用 JButton 类添加按钮组件，并添加事件监听器 ActionListener 和重写 actionPerformed 来实时监听按钮的操作。利用 UIManger 类设置界面，这个类就是 Swing 界面管理的核心，管理 Swing 的小应用程序以及应用程序样式的状态。

【程序实现】

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ButtonDemo {
    private static String labelPrefix = "Number of button clicks: ";
    private int numClicks = 0;

    public Component createComponents() {
        final JLabel label = new JLabel(labelPrefix + "0    ");
        JButton button = new JButton("I'm a Swing button!");
        button.setMnemonic(KeyEvent.VK_I); //设置快捷键为 Alt+I
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                numClicks++;
                label.setText(labelPrefix + numClicks);
```

```
            }
        });
        label.setLabelFor(button); //为即将获得焦点的组件添加描述性信息
        /* 在中间容器和它的组件之间设置空白边框 */
        JPanel pane = new JPanel();
        pane.setBorder(BorderFactory.createEmptyBorder(
                                    30, //上边
                                    30, //左边
                                    10, //下边
                                    30) //右边
                                    );
        pane.setLayout(new GridLayout(0, 1));
        pane.add(button);
        pane.add(label);
        return pane;
    }

    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName());
        } catch (Exception e) { }

        //创建顶层容器并加入中间容器
        JFrame frame = new JFrame("SwingApplication");
        ButtonDemo app = new ButtonDemo();
        Component contents = app.createComponents();
        frame.getContentPane().add(contents, BorderLayout.CENTER);

        //完成 JFrame 的创建并显示
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        frame.pack();
        frame.setVisible(true);
    }
}
```

【运行结果】

运行结果如图 5-3 所示。

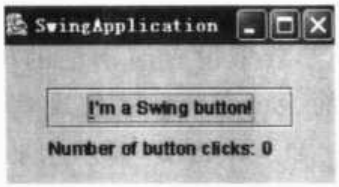


图 5-3

在 button 上每按一次，或者按下 Alt+I，Number of button clicks 就增加 1。

【分析与总结】

当按钮被按下时，通过事件监听器中，numClicks 就会加 1，并重新设置 label 的文本值。在创建内容窗格 pane 时，先设定空白边框，也就是 button 周围的空白位置，然后用到了版面布局管理 GridLayout，从而建立一个组件表格，而且当组件被加入时，会依序由左至右、由上至下摆放在每个格子里。需要在构造函数中指定行数和列数；各行各列的大小是相同的。new GridLayout(0,1)表明行数不限，可以摆放任意个组件，而每一行却只有一个列，即每一行只能摆放一个组件。

在 main()函数里面，用到了 UIManager 类，Swing 使用一种可插式外观和感觉

（Pluggable Look-and-Feel，PLAF）体系结构。开发者不必针对不同组件为颜色和字体之类的设置进行硬编码；取而代之的是，组件将从用户界面管理器（User Interface（UI）Manager）请求这些设置。作为开发者，告诉 UI 管理器要让用户看到什么样的界面；可以选择 Windows、Motif 或 Metal 样式等。UI 管理器接下来就可以实际告诉每个组件应该如何显示它自身。对于像按钮这样的组件来说，前景颜色是通过设置 Button.foreground 属性来控制的，如下所示：

```
UIManager.put("Button.foreground", Color.red);
```

在这里，Button.foreground 为用户界面属性名，而 Color.red 代表特定的设置。改变了设置之后，所有新的按钮的前景颜色都将是红色（也有方法可以改变以前创建的按钮的前景颜色）。虽然不同的预置外观和感觉为这些设置提供了缺省值，还可以重设它们。

getCrossPlatformLookAndFeelClassName()方法则返回缺省的实现了跨平台的 Look and Feel——即 Java Look and Feel（JLF）。

setMnemonic()方法为组件设置快捷键。

setLabelFor()方法可将标签与其他组件相关联，是提供对将要接收焦点的组件的描述。

实例 52 JScrollPane 示例

本实例演示 JScrollPane 的应用。

【编程要点】

JScrollPane 构造函数：

JScrollPane()：建立一个空的 JScrollPane 对象。

JScrollPane(Component view)：建立一个新的 JScrollPane 对象，当组件内容大于显示区域时会自动产生滚动轴。

JScrollPane(Component view,int vsbPolicy,int hsbPllicy)：建立一新的 JScrollPane 对象，里面含有显示组件，并设置滚动轴出现时机。

JScrollPane(int vsbPolicy,int hsbPolicy)：建立一个新的 JScrollPane 对象，里面不含有显示组件，但设置滚动轴出现时机。

JScrollPane 或利用下面这些参数来设置滚动轴的出现的时机，这些参数是定义在 ScrollPaneConstants interface 中，而 JScrollPane 类实现此界面，因此也就能使用这些参数：

HORIZONTAL_SCROLLBAR_ALAWAYS：显示水平滚动轴。

HORIZONTAL_SCROLLBAR_AS_NEEDED：当组件内容水平区域大于显示区域时出现水平滚动轴。

HORIZONTAL_SCROLLBAR_NEVER：不显示水平滚动轴。

VERTICAL_SCROLLBAR_ALWAYS：显示垂直滚动轴。

VERTICAL_SCROLLBAR_AS_NEEDED：当组件内容垂直区域大于显示区域时出现垂直滚动轴。

VERTICAL_SCROLLBAR_NEVER：不显示垂直滚动轴。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
```

```
public class JScrollPaneDemo implements ActionListener {
    private JScrollPane scrollPane;

    public JScrollPaneDemo() {
        JFrame f = new JFrame("JScrollPaneDemo");
        Container contentPane = f.getContentPane();
        JLabel label1 = new JLabel(new ImageIcon(".\\icons\\Hill.jpg"));
        JPanel panel1 = new JPanel();
        panel1.add(label1);
        scrollPane = new JScrollPane();
        //设置窗口显示的内容窗格为 panel1
        scrollPane.setViewportView(panel1);
        //设置水平与垂直表头
        scrollPane.setColumnHeaderView(new JLabel("水平表头"));
        scrollPane.setRowHeaderView(new JLabel("垂直表头"));
        //设置 scrollPane 的边框凹陷立体边框。
        scrollPane.setViewportBorder(BorderFactory
                .createBevelBorder(BevelBorder.LOWERED));
        //设置 scrollPane 的边角图案。
        scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER, new JLabel(
                new ImageIcon(".\\icons\\Sunset.jpg")));
        scrollPane.setCorner(JScrollPane.UPPER_RIGHT_CORNER, new JLabel(
                new ImageIcon(".\\icons\\Sunset.jpg")));
        JPanel panel2 = new JPanel(new GridLayout(3, 1));
        JButton b = new JButton("显示水平滚动轴");
        b.addActionListener(this);
        panel2.add(b);
        b = new JButton("不要显示水平滚动轴");
        b.addActionListener(this);
        panel2.add(b);
        b = new JButton("适时显示水平滚动轴");
        b.addActionListener(this);
        panel2.add(b);
        contentPane.add(panel2, BorderLayout.WEST);
        contentPane.add(scrollPane, BorderLayout.CENTER);
        f.setSize(new Dimension(350, 220));
        f.show();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("显示水平滚动轴")) {
            scrollPane.setHorizontalScrollBarPolicy
                (JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
        }
        if (e.getActionCommand().equals("不要显示水平滚动轴")) {
            scrollPane.setHorizontalScrollBarPolicy
                (JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        }
        if (e.getActionCommand().equals("适时显示水平滚动轴")) {
            scrollPane.setHorizontalScrollBarPolicy
                (JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        }
        scrollPane.revalidate();//重新显示 JScrollPane 形状
    }
    public static void main(String[] args) {
        new JScrollPaneDemo();
    }
}
```

【运行结果】

运行结果如图 5-4 所示。

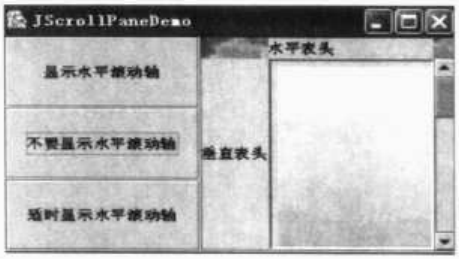


图 5-4

【分析与总结】

本例通过三个按钮触发事件来改变 JScrollPane 的显示方式，由于三个按钮使用的都是同一个监听器，所以在 actionPerformed()中要通过判断事件触发来源而进行不同的动作。

这里还用到了 JPanel 类，Panel 类是 java 中时常用到的容器之一，Panel 除了可以让组件加入外，有效的利用 Panel 可以使版面管理更为容易。Swing 的 JPanel 支持 double buffering 的功能，使得 JPanel 在处理动画上更为流畅，较不会有画面闪烁的情况发生。作为中间容器更加增加了 Swing 窗口设计的层次性。

此外，值得一提的是，在设置 JScrollPane 的边框时用到了 Border 类，Border 类是应用在描绘组件的边界，Border 本身是一个 interface，里面定义了 3 个方法，为 getBorderInsets()、isBorderOpaque()与 paintBorder()。若想使用 Border 类来绘制窗口边界，必须先实现（implements）这 3 个方法。Java 本身提供了另一个类，它已经实现了 Border 类所有的方法，且提供许多不同的边界样式供用户使用，用户只需要选择到底要用哪个样式即可，不需要理会如何去画出这个边界，因为这个类已经实现了，这个类就是 BorderFactorycreat。它提供的方法之一 BevelBorder(int type)可以建立一个立体的边界，并由参数 type 指定为凹陷或突起，type 可为 BevelBorder.LOWERED 表示凹陷，或是 BevelBorder.RAISED 表示突起。

实例 53 JCheckBox 实例

本实例演示 JcheckBox 的应用。

【编程要点】

JCheckBox 构造函数：

JCheckBox()：建立一个新的 JChcekBox。

JCheckBox(Icon icon)：建立一个有图像但没有文字的 JcheckBox。

JCheckBox(Icon icon,boolean selected)：建立一个有图像但没有文字的 JcheckBox，且设置其初始状态（有无被选取）。

JCheckBox(String text)：建立一个有文字的 JcheckBox。

JCheckBox(String text,boolean selected)：建立一个有文字的 JcheckBox，且设置其初始状态（有无被选取）。

JCheckBox(String text,Icon icon)：建立一个有文字且有图像的 JcheckBox，初始状态为无被选取。

JCheckBox(String text,Icon icon,boolean selected)：建立一个有文字且有图像的

JcheckBox，且设置其初始状态（有无被选取）。当 JCheckBox 中的选项被选取或取消时，它会触发 ItemEvent 的事件，ItemEvent 这个类共提供了 4 种方法可以使用，分别是 getItem()、getItemSelectable()、getStateChange()、paramString()。getItem()与 paramString()方法会返回一些这个 JCheckBox 的状态值。一般较少用到这两个方法。

getItemSelectable()相当于 getSource()方法，一样都是返回触发事件的组件，用来判断是哪个组件产生事件。getSource()方法是 EventObject 类所提供，而所有事件类都会继承这个类，因此所有的事件均能用 getSource()方法来判断到底是哪个组件触发了事件。

最后 getStateChange()方法会返回此组件到底有没有被选取。这个方法会返回一个整数值。而我们可以用 ItemEvent 所提供的类变量；若被选取则返回 SELECTED，若没有被选取则返回 DESELECTED。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JCheckBoxDemo implements ItemListener {
    JFrame f = null;
    JCheckBox c4 = null;
    JCheckBox c5 = null;

    JCheckBoxDemo() {
        f = new JFrame("JCheckBox");
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new GridLayout(2, 1));
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(1, 3));
        //建立一个标题边界，并指定标题名称，其他为默认值
        p1.setBorder(BorderFactory.
                createTitledBorder("您最喜欢哪一家速食店呢？"));
        JCheckBox c1 = new JCheckBox("麦当劳");
        JCheckBox c2 = new JCheckBox("肯德鸡");
        JCheckBox c3 = new JCheckBox("21 世纪");
        p1.add(c1);
        p1.add(c2);
        p1.add(c3);
        JPanel p2 = new JPanel();
        p2.setLayout(new GridLayout(2, 1));
        p2.setBorder(BorderFactory.
                createTitledBorder("您喜欢哪种程序语言，喜欢的请打勾："));
        c4 = new JCheckBox("JAVA", new ImageIcon(".\\icons\\x.gif"));
        c5 = new JCheckBox("C++", new ImageIcon(".\\icons\\x.gif"));
        c4.addItemListener(this);
        c5.addItemListener(this);
        p2.add(c4);
        p2.add(c5);
        contentPane.add(p1);
        contentPane.add(p2);
        f.pack();
        f.show();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public void itemStateChanged(ItemEvent e) {
        if (e.getStateChange() == e.SELECTED) {
            if (e.getSource() == c4)
                c4.setIcon(new ImageIcon(".\\icons\\r.gif"));
```

```
                if (e.getSource() == c5)
                    c5.setIcon(new ImageIcon(".\\icons\\r.gif"));

            } else {
                if (e.getSource() == c4)
                    c4.setIcon(new ImageIcon(".\\icons\\x.gif"));
                if (e.getSource() == c5)
                    c5.setIcon(new ImageIcon(".\\icons\\x.gif"));
            }
        }

        public static void main(String args[]) {
            new JCheckBoxDemo();
        }
}
```

【运行结果】

运行结果如图 5-5 所示。

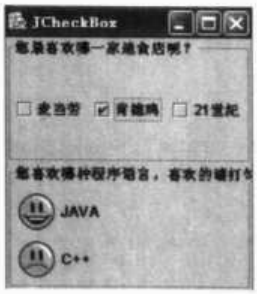


图 5-5

选中复选框 JAVA 或 C++时，图标都会改变。

【分析与总结】

本程序建立了两个中间容器 JPanel，p1 和 p2，它们分别包含组件第一组的复选框和第二组的复选框，然后再将 p1 和 p2 加到主窗口中。在 itemStateChanged()中，通过 getStateChange()来判断是否是属于 select 事件，然后又通过 getSource()判断事件的源头，从而调用 setIcon()来改变复选框的图标。

实例 54 JList 示例

本实例演示 JList 的使用。

【编程要点】

JList 与 JCheckBox 有点相似，都可让程序员选择一到多个选项，不同的是 JList 的选项方式是整列选取。下面来看 JList 所提供的构造函数，它能迅速建立 JList 对象。

JList 构造函数：

JList()：建立一个新的 JList 组件。

JList(ListModel dataModel)：利用 ListModel 建立一个新的 JList 组件。

JList(Object[] listData)：利用 Array 对象建立一个新的 JList 组件。

JList(Vector listData)：利用 Vector 对象建立一个新的 JList 组件。

JList 的事件处理一般可分为两种：一种是取得用户选取的项目；另一种是在 JList 的项目上双击鼠标两次，运行相对应的工作。这里用到第一种事件处理方式：在 JList 类中有 addListSelectionListener()方法，可以检测用户是否对 JList 的选取有任何的改变。ListSelectionListener interface 中只有定义一个方法，那就是 valueChanged

(ListSelectionEvent e)，必须实现这个方法，才能在用户改变选取值时取得用户最后的选取状态，由于 ListSelectionEvent 是 swing 的事件，不是 awt 的事件，因此必须 import。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class JListDemo implements ListSelectionListener {
    JList list = null;
    JLabel label = null;
    String[] s = { "美国", "中国", "英国", "法国", "意大利", "澳洲", "韩国" };

    public JListDemo() {
        JFrame f = new JFrame("JList");
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new BorderLayout());
        label = new JLabel();
        list = new JList(s);
        list.setVisibleRowCount(5); //设定列表方框的栏数
        list.setBorder(BorderFactory
                .createTitledBorder("您最喜欢到哪个国家玩呢? "));
        list.addListSelectionListener(this);
        contentPane.add(label, BorderLayout.NORTH);
        //给列表方框添加滚动栏
        contentPane.add(new JScrollPane(list), BorderLayout.CENTER);
        f.pack();
        f.show();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
    public void valueChanged(ListSelectionEvent e) {
        int tmp = 0;
        String stmp = "您目前选取: ";
      //利用 JList 类所提供的 getSelectedIndices()方法可得到用户所选取的所有项目
        int[] index = list.getSelectedIndices();
      //index 值，这些 index 值由一个 int array 返回
        for (int i = 0; i < index.length; i++)
        {
            tmp = index[i];
            stmp = stmp + s[tmp] + "  ";
        }
        label.setText(stmp);
    }
    public static void main(String args[]) {
        new JListDemo();
    }
}
```

【运行结果】

运行结果如图 5-6 所示。



图 5-6

按住 Ctrl 键，再用鼠标单击，可以选择多个项。

【分析与总结】

本程序用一个 String 数组来构造一个 JList，在 valueChange()方法中，通过 getSelectedIndices()方法得到用户所选取的所有项目，并把它们在 Jlabel 中显示出来。

实例 55 JTree 示例

本实例演示 JTree 组件的应用。

【编程要点】

JTree 构造函数：

JTree()：建立一棵系统默认的树。

JTree(Hashtable value)：利用 Hashtable 建立树，不显示 root node（根节点）。

JTree(Object[] value)：利用 Object Array 建立树，不显示 root node。

JTree(TreeModel newModel)：利用 TreeModel 建立树。

JTree(TreeNode root)：利用 TreeNode 建立树。

JTree(TreeNode root,boolean asksAllowsChildren)：利用 TreeNode 建立树，并决定是否允许子节点的存在。

JTree(Vector value)：利用 Vector 建立树，不显示 root node。

JTree 上的每一个节点就代表一个 TreeNode 对象，TreeNode 本身是一个 Interface，里面定义了 7 个有关节点的方法，例如判断是否为树叶节点（isNodeChild()）、有几个子节点（getChildCount()）、父节点为何（getparent()）等等、这些方法的定义可以在 javax.swing.tree 的 package 中找到。在实际的应用上，一般不会直接实现此界面，而是采用 java 所提供的 DefaultMutableTreeMode 类，此类是实现 MutableTreeNode 界面而来，并提供了其他许多实用的方法。MutableTreeNode 本身也是一个 Interface，且继承了 TreeNode 界面，此类主要是定义一些节点的处理方式，例如新增节点（insert()）、删除节点（remove()）、设置节点（setUserObject()）等。整个关系如下：

TreeNode----extends--->MutableTreeNode---implements---DefaultMutableTreeNode

DefaultMutableTreeNode 的构造函数：

DefaultMutableTreeNode()：建立空的 DefaultMutableTreeNode 对象。

DefaultMutableTreeNode(Object userObject)：建立 DefaultMutableTreeNode 对象，节点为 userObject 对象。

DefaultMutableTreeNode(Object userObject,Boolean allowsChildren)

建立 DefaultMutableTreeNode 对象，节点为 userObject 对象并决定此节点是否允许具有子节点。

【程序实现】

```
//TreeNodeDemo.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.tree.*;

public class JTreeDemo extends JFrame {
    protected JTree m_tree;
```

```
        protected JTextField m_display;

        public JTreeDemo() {
            super("SNMP Tree [OID]");
          //创建 5 个树的用户对象引用
          Object[] nodes = new Object[5];
            //创建树的根节点（第一层节点）
            DefaultMutableTreeNode top = new DefaultMutableTreeNode(new
OidNode(1,"ISO"));
            DefaultMutableTreeNode parent = top;
            //创建树根节点的子节点（第二层节点）
            DefaultMutableTreeNode node = new DefaultMutableTreeNode(new
OidNode(0,"standard"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(2, "member-body"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(3, "org"));
            parent.add(node);
            parent = node;
            //创建第二层节点的子节点（第三层节点）
            node = new DefaultMutableTreeNode(new OidNode(6, "dod"));
            parent.add(node);
            parent = node;
            //创建第三层节点的子节点（第四层节点）
            node = new DefaultMutableTreeNode(new OidNode(1, "internet"));
            parent.add(node);
            parent = node;
            //创建第四层节点的子节点（第五层节点）
            node = new DefaultMutableTreeNode(new OidNode(1, "directory"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(2, "mgmt"));
            parent.add(node);
            node.add(new DefaultMutableTreeNode(new OidNode(1, "mib-2")));
            node = new DefaultMutableTreeNode(new OidNode(3, "experimental"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(4, "private"));
            node.add(new DefaultMutableTreeNode(new OidNode(1, "enterprises")));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(5, "security"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(6, "snmpV2"));
            parent.add(node);
            node = new DefaultMutableTreeNode(new OidNode(7, "mail"));
            parent.add(node);
            //以树的根节点构造整个树
          m_tree =new JTree(top)
          {
              //设定 JTree 的工具提示
              public String getToolTipText(MouseEvent e) {
                    //获取鼠标在树组件中选择位置的树的路径
                    TreePath path = m_tree.getPathForLocation(e.getX(), e.getY());
                    //如果鼠标没有选择树节点上，则直接返回
                    if (path == null)
                        return null;
                    //获取当前树路径的存储的树节点对象的数组（从根节点到当前树节点）
                    Object[] nodes = path.getPath();
                    String oid = "";
                    for (int k = 0; k < nodes.length; k++) {
                        //强制转化成树节点
                        DefaultMutableTreeNode treenode = (DefaultMutableTreeNode)
nodes[k];
                        //从树节点中获取其用户对象（OidNode 的实例）
                        OidNode nd = (OidNode) treenode.getUserObject();
                        //把用户对象的对象标识进行累积
                        oid += "." + nd.getId();
                    }
                    return oid;
```

```
            }
        };
        ToolTipManager.sharedInstance().registerComponent(m_tree);

        //显示 JTree 组件的句柄
        m_tree.setShowsRootHandles(true);
        //使 JTree 具有类似 Windows 文件管理器的直角连接线
        m_tree.putClientProperty("JTree.lineStyle", "Angled");
        //设置树的树节点是不可编辑的
        m_tree.setEditable(false);
        //添加监视树的鼠标事件
        m_tree.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                tvwTreeView_mouseClicked(e);
            }
        });
        //把树组件加入到滚动面板中
        JScrollPane s = new JScrollPane(m_tree);
        getContentPane().add(s, BorderLayout.CENTER);
        m_display = new JTextField();
        m_display.setEditable(false);
        getContentPane().add(m_display, BorderLayout.SOUTH);
        //显示框架窗口
        setSize(400, 300);
        setVisible(true);
    }

    //处理树组件的鼠标单击事件
    private void tvwTreeView_mouseClicked(MouseEvent e) {
        //获取鼠标在树组件中选择位置的树的路径
        TreePath path = m_tree.getPathForLocation(e.getX(), e.getY());
        //如果鼠标没有选择树节点上，则直接返回
        if (path == null)
            return;
        //获取当前树路径的存储的树节点对象的数组（从根节点到当前树节点）
        Object[] nodes = path.getPath();
        String oid = "";
        for (int k = 0; k < nodes.length; k++) {
            //强制转化成树节点
            DefaultMutableTreeNode node = (DefaultMutableTreeNode) nodes[k];
            //从树节点中获取其用户对象（OidNode 的实例）
            OidNode nd = (OidNode) node.getUserObject();
            //把用户对象的对象名称进行累积
            oid += "." + nd.getName();
        }
        m_display.setText(oid);
    }

    public static void main(String argv[]) {
        JTreeDemo frame = new JTreeDemo();
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}

//树的用户对象类（UserObject），用于封装 SNMP 的 OID
class OidNode {
    //对象标识
    private int m_id;
    //对象名称
    private String m_name;
    //构造方法
    public OidNode(int id, String name) {
```

```
        m_id = id;
        m_name = name;
    }
    //获取对象标识
    public int getId() {
        return m_id;
    }
    //得到对象名称
    public String getName() {
        return m_name;
    }
    //用户对象的字符串化，这个方法重写了 Object.toString()
    public String toString() {
        return m_name;
    }
}
```

【运行结果】

运行结果如图 5-7 所示。

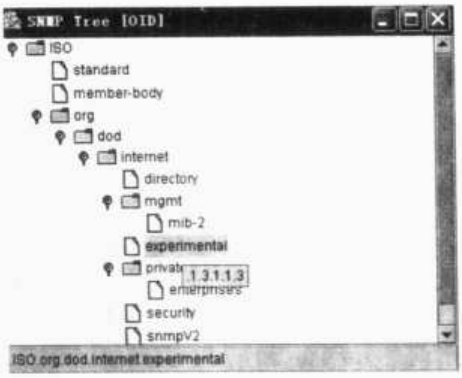


图 5-7

鼠标选中 experimental，在最下方的 JTextFeild 中显示该项的全路径。

【分析与总结】

本程序添加了监视树的鼠标事件，在 tvwTreeView_mouseClicked()事件处理方法中，先通过 getPath()得到鼠标单击的节点的当前路径，然后将路径转化为各个节点，最后将各个节点名称连接起来并在 JTextField 中显示。

为了改变 JTree 的外观，可以使用 JComponent 所提供的 putClientProperty(Object key,Object value)方法来设置 Java 默认的 JTree 外观，设置方式共有 3 种：

（1）tree.putClientProperty("JTree.lineStyle","None")：Java 默认值。

（2）tree.putClientProperty("JTree.lineStyle","Horizontal")：使 JTree 的文件夹间具有水平分隔线。

（3）tree.putClientProperty("JTree.lineStyle","Angled")：使 JTree 具有类似 Windows 文件管理器的直角连接线。

本程序还用到了 ToolTip，几乎所有的会被用来建构用户界面的 classes 都继承自 JComponent，而 JComponent 拥有一个 getToolTipText(MouseEvent e) 和 setToolTipText(String)，所以，所有可被放在表单（forms）上的东西几乎都可以调用这些方法，第一个方法用来产生工具文本，并返回 String，第二个方法把 String 添加到工具提示中。最后 ToolTipManager.sharedInstance().registerComponent(m_tree)将工具提示注册到 m_tree 中。

实例 56 JTable 示例

本实例演示如何使用 JTable 及怎样处理 JTable 的触发事件。

【编程要点】

JTabel 构造函数：

JTable()：建立一个新的 JTables，并使用系统默认的 Model。

JTable(int numRows,int numColumns)：建立一个具有 numRows 行，numColumns 列的空表格，使用的是 DefaultTableModel。

JTable(Object[][] rowData,Object[][] columnNames)：建立一个显示二维数组数据的表格，且可以显示列的名称。

JTable(TableModel dm)：建立一个 JTable，有默认的字段模式以及选择模式，并设置数据模式。

JTable(TableModel dm,TableColumnModel cm)：建立一个 JTable，设置数据模式与字段模式，并有默认的选择模式。

JTable(TableModel dm,TableColumnModel cm,ListSelectionModel sm)：建立一个 JTable，设置数据模式、字段模式与选择模式。

JTable(Vector rowData,Vector columnNames)：建立一个以 Vector 为输入来源的数据表格，可显示行的名称。

TableModel 类本身是一个 interface，在这个 interface 里面定义了若干的方法：包括了存取表格字段（cell）的内容、计算表格的列数等等的基本存取操作，让设计者可以简单地利用 TableModel 来实现他所想要的表格。TableModel 界面是放在 javax.swing.table package 中，这个 package 定义了许多 JTable 会用到的各种 Model。

AbstractTableModel 是一个抽象类，这个类帮我们实现大部份的 TableModel 方法，因此利用这个抽象类就可以设计出不同格式的表格。

在 JTable 的事件大致均针对表格内容的异操作处理，包括字段内容改变，列数增加或减少，行数增加或减少，或是表格的结构改变等等。这些事件称为 TableModelEvent 事件。要处理 TableModelEvent 事件必须实现 TableModelListener 界面，此界面定义了一个方法，那就是 tableChanged()，为了处理这些事件的种种情况，在 AbstractTableModel 类中提供了下列方法来提示 TableModelListener 表格内容已经改动了，如下所示：

（1）fireTableCellUpdated()：发出表格中的某一个字段已经更改的事件信息。

（2）fireTableChanged()：发出表格已经改动的事件信息。

（3）fireTableDataChanged()：发出表格中的某字段已经更改的事件信息。

（4）fireTableRowsDeleted()：发出表格中的某几行已经删除的事件信息。

（5）fireTableRowsInserted()：发出表格中的已经新增某几行的事件信息。

（6）fireTableRowsUpdated()：发出表格中的某几行已经修改的事件信息。

（7）fireTableStructureChanged()：发出表格结构已经改变的事件信息，这里指的结构改变可能包括表格的列数已经改变。

在知道了在表格中可能发生的事件后，如何拦截这些事件的信息呢？在 AbstractTableModel 类中提供了一个注册 listener 的方法 addTableModelListener()。在加入

TableModelListener 之后，就可以依照不同的事件做不同的处理了。

【程序实现】

```
import javax.swing.table.AbstractTableModel;
import javax.swing.event.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class JTableDemo implements TableModelListener
{
    JTable table = null;
    MyTable mt = null;
    JLabel label = null; //显示修改字段位置

    public JTableDemo() {
          JFrame f = new JFrame();
          mt=new MyTable();
          //增加 JTable 事件监听器
          mt.addTableModelListener(this);
          table=new JTable(mt);
          //创建组合框
          JComboBox c = new JComboBox();
          //给组合框增加条目
          c.addItem("广州");
          c.addItem("上海");
          c.addItem("北京");
          //利用组合框作为 table 的第一列值的选项
          table.getColumnModel().
          getColumn(1).setCellEditor(new DefaultCellEditor(c));
          //设定 table 的显示的大小
          table.setPreferredScrollableViewportSize(new Dimension(550, 40));
          //给 table 增加滚动条
        JScrollPane s = new JScrollPane(table);
        label = new JLabel("修改字段位置: ");
        f.getContentPane().add(s, BorderLayout.CENTER);
        f.getContentPane().add(label, BorderLayout.SOUTH);
         f.setTitle("TableEventHandle");
        f.pack();
        f.setVisible(true);

        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public void tableChanged(TableModelEvent e)
    {
     int row = e.getFirstRow();  //得到被修改的行(第一个)的行数
        int column = e.getColumn(); //得到被修改的列数
        label.setText("修改字段位置: "+(row+1)+" 行 "+(column+1)+" 列");
        //得到第 row 行第 6 列的值，即看看是否作弊
        boolean cheat =((Boolean)(mt.getValueAt(row,6))).booleanValue();
        int grade1=((Integer)(mt.getValueAt(row,2))).intValue();
        int grade2=((Integer)(mt.getValueAt(row,3))).intValue();
         int total = grade1+grade2;
     if(cheat)
     {   //如果作弊，则总分重设
          if(total > 120)
              mt.mySetValueAt(new Integer(119),row,4);
          else
              mt.mySetValueAt(new Integer(total),row,4);
          mt.mySetValueAt(new Boolean(false),row,5);
```

```
        }
        else
        {
            if(total > 120)    //判断该学生是否及格
                mt.mySetValueAt(new Boolean(true),row,5);
            else
                mt.mySetValueAt(new Boolean(false),row,5);

            mt.mySetValueAt(new Integer(total),row,4);
        }
        table.repaint();
       }

       public static void main(String args[]) {
          new JTableDemo();
       }
}

class MyTable extends AbstractTableModel {
    Object[][] p = {
    {"小明", "广州",new Integer(66), new Integer(32), new Integer(98),
     new Boolean(false),new Boolean(false)},
    {"小军", "上海",new Integer(85), new Integer(69), new Integer(154),
     new Boolean(true),new Boolean(false)},
     {"小东", "北京",new Integer(88), new Integer(70), new Integer(158),
     new Boolean(false),new Boolean(false)}};
    String[] n = {"姓名","居住地","语文","数学","总分","及格","作弊"};

    //返回总列数
    public int getColumnCount() {
        return n.length;
    }

    //返回总行数
    public int getRowCount() {
        return p.length;
    }

    //返回列名
    public String getColumnName(int col) {
        return n[col];
    }

    //返回 table 的一个字段，若没有设置列标题则返回默认值，依次为 A,B,C,...Z,AA,AB,..;
    //若无此 column,则返回一个空的 String
    public Object getValueAt(int row, int col) {
        return p[row][col];
    }

    //返回字段数据类型的类名称
     public Class getColumnClass(int c) {
          return getValueAt(0, c).getClass();
     }

      //返回指定字段是否可修改
      public boolean isCellEditable(int rowIndex, int columnIndex) {
          return true;
      }

      //给指定字段赋值，并通知 table 已修改
      public void setValueAt(Object value, int row, int col) {
          p[row][col] = value;
          fireTableCellUpdated(row, col);
      }

      //自定义修改某指定字段
     public void mySetValueAt(Object value, int row, int col) {
```

```
            p[row][col] = value;
        }
    }
```

【运行结果】

运行结果如图 5-8 所示。

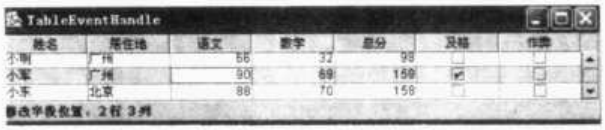


图 5-8

【分析与总结】

本例通过继承 AbstractTableModel 类创建了 MyTable 类来作为 JTable 的初始化参数，所有获取和修改表格字段信息的函数都在 MyTable 类中覆写了。然后在主 Frame 中添加监听器 addTableModelListener()并覆写 tableChanged()来捕获表格修改的事件从而作相应的处理。

从运行结果可以看出所有 String 类型都靠右显示，所有数据类型都靠左显示，而 Boolean 类型是以 CheckBox 的形式显示的，这是因为 MyTable 类中覆写了 getColumnClass()方法返回字段的类型，通过这个方法可以让读者分辨出表格中每一行的数据类型，并将此类型作适当的显示。

实例 57 Java Application 启动画面制作

本实例制作一个程序启动时的画面。

【编程要点】

创建一个 JWindow 类的子类，并生成一个对象 windowSplash，在初始化时显示。用 windowSplash.toFont()使其处于最前，初始化结束后使用 windowSplash.dispose()关闭它。本实例使用两个线程分别显示启动窗口和主窗口。

【程序实现】

```
import javax.swing.*;
import java.awt.*;
import java.net.*;

public class JSplashWindow extends JWindow implements Runnable {
    Thread splashThread = null;

    public JSplashWindow() {
        setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        JPanel splash = new JPanel(new BorderLayout());
        //获得图片资源的绝对路径
        URL url = getClass().getResource("/images/tu.gif");
        if (url != null) { //图片嵌入 JLabel 中显示
            splash.add(new JLabel(new ImageIcon(url)), BorderLayout.CENTER);
        }
        setContentPane(splash);  //给启动窗口设置内容窗格
        Dimension screen = getToolkit().getScreenSize(); //获得屏幕的大小
        pack();
        //设定启动窗口在屏幕中的位置
        setLocation((screen.width - getSize().width) / 2,
                (screen.height - getSize().height) / 2);
    }

    public void start() {
```

```
        this.toFront();  //将启动窗口放在当前位置
        splashThread = new Thread(this);
        splashThread.start();
    }

    public void run() {
        try {
            show();
            Thread.sleep(5000);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        this.dispose();
    }

    static void showFrame(String title) {
        JFrame frame = new JFrame(title);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //将画窗置于屏幕中央
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
                (screenSize.height - frameSize.height) / 2);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        showFrame("Demo splash window"); //显示画窗
        JSplashWindow splash = new JSplashWindow();
        splash.start(); //显示启动窗口,并维持一定时间
    }
}
```

【运行结果】

运行结果如图 5-9 所示。

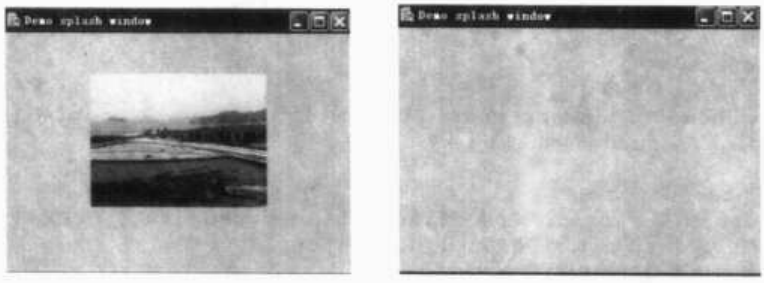


图 5-9

【分析与总结】

这里用到了两个线程，一个是显示主窗口，一个显示启动窗口。由于 showFrame()是 static 函数，所以当 splash 线程 start 以后，进入 run()，调用 show()显示启动窗口，然后让线程休眠 5 秒，最后调用 dispose()清除屏幕资源，因此就只看到了原来的 Frame 了。

实例 58　带进度条的启动窗口

本实例演示如何创建一个带进度条显示的启动窗口。

【编程要点】

在安装一个新软件时，系统会告知目前软件安装的进度如何，以便让用户掌握程序的运行情况，同样的，若设计的程序所需要的运行超过 2 秒以上，应该显示程序正在运行中的图标，或直接显示程序运行的进度，这样就能让用户清楚知道程序到底是终止了还是继续运行，在 swing 中，JProgressBar 组件提供了类似这样的功能，它可以简单地输出进度的变化情况，让程序员想要提供进度信息时，不再需要自行绘制绘图组件，只需要使用 JProgressBar 再加上几行程序设置就可以了。本例实际上就是在上例程序的基础上再加上进度条组件。常用的函数有：

JProgressBar progress = new JProgressBar(int startValue, int endValue)：构造函数，其中 startValue 为进度条的开始值，endValue 为进度条的结束值。

progress.setValue(int Value)：设定进度条的值。

progress.getValue()：获得进度条当前的值。

一般通过 getValue()方法和 setValue()方法来控制进度条的进度。

【程序实现】

```
//JSplashWindowPlus.java

import javax.swing.*;
import java.awt.*;
import java.net.*;

public class JSplashWindowPlus extends JWindow implements Runnable {
    Thread splashThread = null;
    private JProgressBar progress;

    public JSplashWindowPlus() {
        setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        JPanel splash = new JPanel(new BorderLayout());
        URL url = getClass().getResource("/images/tu.gif");
        if (url != null) {
            splash.add(new JButton(new ImageIcon(url)), BorderLayout.CENTER);
        }
        progress = new JProgressBar(1, 100);
        //设置进度条启动时可显示字符串
        progress.setStringPainted(true);
        //设置进度条边框
        progress.setBorderPainted(true);
        progress.setString("Program is Now Loading...");
        progress.setBackground(Color.white);
        //将进度条加入内容窗格中
        splash.add(progress, BorderLayout.SOUTH);
        setContentPane(splash);
        Dimension screen = getToolkit().getScreenSize();
        pack();
        //设置显示位置
        setLocation((screen.width - getSize().width) / 2,
                (screen.height - getSize().height) / 2);
    }

    public void start() {
        this.toFront();
        splashThread = new Thread(this);
        splashThread.start();
    }

    public void run() {
        show();
        try {
            for (int i = 0; i < 100; i++) {
```

```
                Thread.sleep(100);
                progress.setValue(progress.getValue() + 1);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        dispose();
    }

    static void showFrame(String title) {
        JFrame frame = new JFrame(title);
        frame.setSize(640, 480);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //将窗口置于屏幕中央
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
                (screenSize.height - frameSize.height) / 2);

        try {
            Thread.sleep(10000);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        frame.setVisible(true);

    }

    public static void main(String[] args) {
        JSplashWindowPlus splash = new JSplashWindowPlus();
        splash.start();
        showFrame("Splash window demo");
    }
}
```

【运行结果】

运行结果如图 5-10 所示。



图 5-10

当用户启动程序时，最初会呈现图 5-10，并且进度杆线就会开始向右移动，并显示 Program is Now Loading 的信息。

【分析与总结】

这里是通过如下代码设置进度条的值，每经过 100 毫秒，进度条值加 1：

```
for (int i = 0; i < 100; i++) {
    Thread.sleep(100);
    progress.setValue(progress.getValue() + 1);
}
```

另外，每当 JProgressBar 的值改变一次（利用 setValue()方法），就会触发一次

ChangeEvent 事件，如果要处理这个事件，必须实现 ChangeListener 界面所定义的 stateChanged()方法，有兴趣的读者自行实现。

实例 59　文件选择对话框

本实例演示文件选择对话框的实现。

【编程要点】

用 FileDialog 类创建一个文件对话框，FileDialog 具有以下的构造函数：

```
FileDialog(Frame parent, String title);
FileDialog(Frame parent, String title, int how);
FileDialog(Frame parent);
```

其中 parent 指定对话框的所有者，即父窗口；title 指定对话框的标题，忽略则为空；how 为 FileDialog.LOAD，对话框为读选取文件（默认），为 FileDialog.SAVE，为写文件。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class MyFileDialog extends JFrame {
    MyFileDialog(String title) {
        super(title);
        this.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });
    }
}

public class FileDialogDemo {
    public static void main(String args[]) {
        Frame f = new MyFileDialog("File Dialog Demo!");
        f.setVisible(true);
        f.setSize(100, 100);
        FileDialog fd = new FileDialog(f, "File Dialog");
        fd.setVisible(true);
    }
}
```

【运行结果】

运行结果如图 5-11 所示。

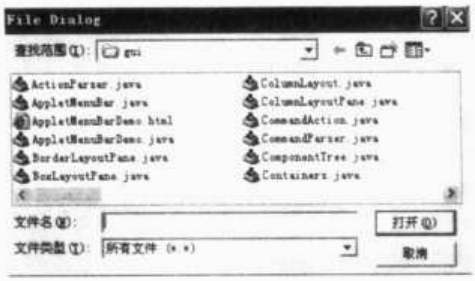


图 5-11

【分析与总结】

文件选择对话框是界面设计中经常要用到的组件，应该好好掌握。

实例 60　菜单

该程序演示一般的菜单的制作和弹出式菜单的制作。

【编程要点】

每一种能够容纳菜单的组件，包括 Japplet、Jframe、Jdialog 及其派生类，都具备有 setJMenuBar()方法，它接受一个 JMenuBar 作为参数（一个组件只允许一个 JMenuBar）。在进行菜单设计时，将 JMenu 加到 JMenuBar 中，将 JMenuItem 加到 Jmenu 中，每个 JmenuItem 都可以拥有连接于其上的 ActionListener，当菜单选项被选取时，即会触发事件。

JPopupMenu 是一种特别形式的 Menu，其性质与 Menu 几乎完全相同，但是 PopupMenu 并不固定在窗口的任何一个位置，而是由鼠标指针和系统判断决定 PopupMenu 要出现在哪里。实现弹出式菜单最直接的方式就是，编写继承自 MouseAdapter 的内隐类（inner class），然后将其对象加到每个你希望具备弹出式行为的组件上。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class MenuDemo extends JApplet {
    JTextField t = new JTextField(15);  //建立单行文本域
    ActionListener al = new ActionListener(){
        public void actionPerformed(ActionEvent e){ //在JTextField中显示菜单标题
            t.setText(((JMenuItem)e.getSource()).getText());
        }
    };
    JMenu[] menus = { new JMenu("文件"),
                      new JMenu("编辑"),
                        new JMenu("帮助")};
    JMenuItem[] items = {
        new JMenuItem("新建"), new JMenuItem("撤销"),
        new JMenuItem("帮助主题"), new JMenuItem("保存"),
        new JMenuItem("剪切"),new JMenuItem("查找"),
        new JMenuItem("退出"),new JMenuItem("复制"),
        new JMenuItem("关于") };
    JPopupMenu popup = new JPopupMenu(); //定义一个弹出式菜单

    public void init(){
        for(int i = 0; i < items.length; i++){
         //为每一个菜单项增加监听器
            items[i].addActionListener(al);
            //将每个菜单项添加到各自的菜单中
            menus[i%3].add(items[i]);
        }
       JMenuBar mb = new JMenuBar();
       for(int i = 0; i < menus.length; i++){
        //将每个菜单添加到菜单控制栏中
           mb.add(menus[i]);
           setJMenuBar(mb);
       }

       Container cp = getContentPane();
       //设定内容窗格的布局控制为FlowLayout格式
       cp.setLayout(new FlowLayout());
       cp.add(t);
       //为弹出式菜单添加菜单项和监视器
       JMenuItem m = new JMenuItem("最小化");
       popup.add(m);
       m.addActionListener(al);
       popup.addSeparator();  //在菜单的末尾添加分隔线
       m = new JMenuItem("最大化");
       popup.add(m);
```

```
        m.addActionListener(al);
        PopupListener pl = new PopupListener();
        addMouseListener(pl);  //将弹出菜单事件添加到鼠标事件处理模式中
    }

    class PopupListener extends MouseAdapter{
        public void mousePressed(MouseEvent e){
            maybeShowPopup(e);
        }

        public void mouseReleased (MouseEvent e){
            maybeShowPopup(e);
        }

        private void maybeShowPopup(MouseEvent e){
            if(e.isPopupTrigger()){
                //判断是否在当前鼠标位置显示该菜单
                popup.show(e.getComponent(),e.getX(),e.getY());
            }
        }
    }

    public static void main(String args[]) {
        JApplet applet = new MenuDemo();
        //创建一个 Frame，将 applet 添加到 frame 中
        JFrame frame = new JFrame("Menu Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(applet);
        applet.init();
        applet.start();
        frame.pack();
        frame.setVisible(true);
    }
}
```

【运行结果】

运行结果如图 5-12 所示。



图 5-12

【分析与总结】

每个 JmenuItem 都必须连接一个 ActionListener；本程序全部使用同一个 ActionListener。

当菜单项被选中时，就会触发文本显示菜单标题。这里弹出式菜单 JPopupMenu 与一般菜单不同的是要为它写一个专门的监听器（程序中的 PopupListener），然后将这个监听器添加到鼠标事件处理模式中。在 PopupListener 中覆写 mousePressed()和 mouseReleased()等鼠标事件来显示弹出式菜单。

还有需要提一下的是，这里用到了布局管理的 FlowLayout，这种布局管理器很单纯地将组件由左至右依次排列在表单上，直到上端空间塞满了才会移到下一列，相同的水平排列方式不断持续进行。

实例 61 对话框

本实例演示 JDialog 的应用。

【编程要点】

JDialog 构造函数：

JDialog()：建立一个 non-modal 的对话框，没有 title 也不属于任何事件窗口组件。

JDialog(Dialog owner)：建立一个属于 Dialog 组件的对话框，为 non-modal 形式，也没有 title。

JDialog(Dialog owner,Boolean modal)：建立一个属于 Dialog 组件的对话框，可决定 modal 形式，但没有 title。

JDialog(Dialog owner,String title)：建立一个属于 Dialog 组件的对话框，为 non-modal 形式，对话框上有 title。

JDialog(Dialog owner,String title,Boolean modal)：建立一个属于 Dialog 组件的对话框，可决定 modal 形式，且对话框上有 title。

JDialog(Frame owner)：建立一个属于 Frame 组件的对话框，为 non-modal 形式，也没有 title。

JDialog(Frame owner,Boolean modal)：建立一个属于 Frame 组件的对话框，可决定 modal 形式，但没有 title。

JDialog(Frame owner,String title)：建立一个属于 Frame 组件的对话框，为 non-modal 形式，对话框上有 title。

JDialog(Frame owner,String title,Boolean modal)：建立一个属于 Frame 组件的对话框，可决定 modal 形式，且对话框上有 title。

上面所说的 modal 是一种对话框操作模式，当 modal 为 true 时，代表用户必须结束对话框才能回到原来所属的窗口。当 modal 为 false 时，代表对话框与所属窗口可以互相切换，彼此之间在操作上没有顺序性。一般而言对话框都会依附在某个窗口上，例如 JFrame 或 JDialog，原因在于对话框通常是一个程序运行的过程中与用户互动的中间过程，在使用 JDialog 上跟 JFrame 非常相似，JDialog 是继承 AWT 的 Dialog 类而来，因此 JDialog 为一个 Heavyweight 组件。要加入组件到 JDialog 上与 JFrame 是一样的，必须先取得 JDialog 的 ContentPane，然后再把组件加到此 ContentPane 中，JDialog 默认的版面管理器是 BorderLayout。

【程序实现】

```
//JDialogDemo.java
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class JDialogDemo implements ActionListener {
    JFrame f = null;

    public JDialogDemo() {
        f = new JFrame("JDialogDemo");
        Container contentPane = f.getContentPane();
        JPanel buttonPanel = new JPanel();
        JButton b = new JButton("借用物品");
        b.addActionListener(this);
        buttonPanel.add(b);
        b = new JButton("离开系统");
        b.addActionListener(this);
        buttonPanel.add(b);
```

```
        buttonPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory
                .createLineBorder(Color.blue, 2), "借用物品系统",
                TitledBorder.CENTER, TitledBorder.TOP));
        contentPane.add(buttonPanel, BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }

    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        if (cmd.equals("借用物品")) {
            new LendingSystem(f);
        } else if (cmd.equals("离开系统")) {
            System.exit(0);
        }
    }

    public static void main(String[] args) {
        SwingUtil.setLookAndFeel();
        new JDialogDemo();
    }
}

class LendingSystem implements ActionListener {
    JTextField staffField, objectField, borrowDateField, returnDateField,
            reasonField;
    JDialog dialog;

    LendingSystem(JFrame f) {
        dialog = new JDialog(f, "借用物品", true);
        GridBagConstraints c;
        int gridx, gridy, gridwidth, gridheight,
            anchor, fill, ipadx, ipady;
        double weightx, weighty;
        Insets inset;

        GridBagLayout gridbag = new GridBagLayout();
        Container dialogPane = dialog.getContentPane();
        dialogPane.setLayout(gridbag);

        JLabel label = new JLabel("员工编号 : ");
        gridx = 0;                              //第 0 列
        gridy = 0;                              //第 0 行
        gridwidth = 1;                          //占一单位宽度
        gridheight = 1;                         //占一单位高度
        weightx = 0;                            //窗口增大时组件宽度增大比率 0
        weighty = 0;                            //窗口增大时组件高度增大比率 0
        anchor = GridBagConstraints.CENTER;     //容器大于组件 size 时将组件置于容器中央
        fill = GridBagConstraints.BOTH;         //窗口拉大时会填满水平与垂直空间
        inset = new Insets(0, 0, 0, 0);         //组件间间距
        ipadx = 0;                              //组件内水平宽度
        ipady = 0;                              //组件内垂直高度
        c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
                weightx, weighty, anchor, fill, inset, ipadx, ipady);
        gridbag.setConstraints(label, c);
        dialogPane.add(label);

        label = new JLabel("借用器具 : ");
        gridx = 3;
        gridy = 0;
        c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
```

```
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(label, c);
dialogPane.add(label);
label = new JLabel("借用日期: ");
gridx = 0;
gridy = 1;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(label, c);
dialogPane.add(label);

label = new JLabel("归还日期: ");
gridx = 3;
gridy = 1;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(label, c);
dialogPane.add(label);

label = new JLabel("借用原因 : ");
gridx = 0;
gridy = 2;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(label, c);
dialogPane.add(label);

staffField = new JTextField();
gridx = 1;
gridy = 0;
gridwidth = 2;
gridheight = 1;
weightx = 1;
weighty = 0;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(staffField, c);
dialogPane.add(staffField);

objectField = new JTextField();
gridx = 4;
gridy = 0;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(objectField, c);
dialogPane.add(objectField);

borrowDateField = new JTextField();
gridx = 1;
gridy = 1;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(borrowDateField, c);
dialogPane.add(borrowDateField);

returnDateField = new JTextField();
gridx = 4;
gridy = 1;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(returnDateField, c);
dialogPane.add(returnDateField);

reasonField = new JTextField();
gridx = 1;
gridy = 2;
gridwidth = 5;
c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
        weightx, weighty, anchor, fill, inset, ipadx, ipady);
gridbag.setConstraints(reasonField, c);
```

```
        dialogPane.add(reasonField);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(1, 2));
        JButton b = new JButton("确定");
        panel.add(b);
        b = new JButton("取消");
        b.addActionListener(this);
        panel.add(b);

        gridx = 0;
        gridy = 3;
        gridwidth = 6;
        weightx = 1;
        weighty = 1;
        c = new GridBagConstraints(gridx, gridy, gridwidth, gridheight,
                weightx, weighty, anchor, fill, inset, ipadx, ipady);
        gridbag.setConstraints(panel, c);
        dialogPane.add(panel);

        dialog.setBounds(200, 150, 400, 130);
        dialog.show();
    }
    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        if (cmd.equals("确定")) {
            //这里可以添加你要处理的动作
            //比如将数据保存到数据库等

        } else if (cmd.equals("取消")) {
            dialog.dispose();
        }
    }
}

class SwingUtil {
    public static final void setLookAndFeel() {
        try {
            Font font = new Font("JFrame", Font.PLAIN, 12);
            Enumeration keys = UIManager.getLookAndFeelDefaults().keys();
            //将所有字体都设置为 font 对象中定义的字体
            while (keys.hasMoreElements()) {
                Object key = keys.nextElement();
                if (UIManager.get(key) instanceof Font) {
                    UIManager.put(key, font);
                }
            }
            //Windows 风格的界面
            UIManager.setLookAndFeel
                ("com.sun.java.java.swing.plaf.windows.WindowsLookAndFeel");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

【运行结果】

运行结果如图 5-13 所示。

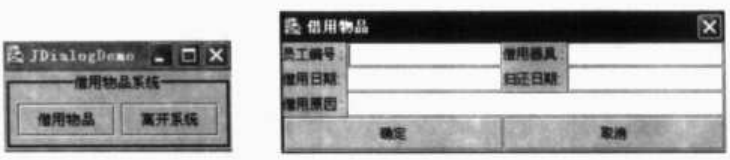


图 5-13

用户在主窗口按下“借用物品”的按钮时，会跳出一个让用户填写窗体的 JDialog 窗

口，用户必须结束此 JDialog 窗口后才能返回主窗口。

【分析与总结】

通过处理按钮事件来调用 JDialog，在 actionPerformed()中通过 getActionCommand()来判断事件来源，从而做出相应的操作。在 JDialog 中也跟 JFrame 类似，可以添加各种组件并进行相应的事件处理。其中在 JDialog 中采用的版面布局是 GridBagLayout，能够提供极佳的控制能力，可以更精确地指定窗口的各个区域的布局方式，以及在窗口大小有所改变时如何重新排列这些区域，它同时也是最复杂的布局控制器。本例还用到了界面风格的设置，在 SwingUtil 类中的 setLookAndFeel，先用 Enumeration（迭代类）列举所有 UIManager 中的 font 对象，并将其改为自定义的 font；然后调用 UIManager 的 setLookAndFeel()方法设置 Windows 风格的界面。

另外还有其他风格，如 com.sun.java.swing.plaf.motif.MotifLookAndFeel 和 javax.swing.plaf.metal，读者不妨一试。

实例 62 计算器

本实例实现了一个简单的计算器，具有整数和小数的加减乘除的功能。

【编程要点】

通过按钮事件来获得用户的输入，再通过计算，将结果输出到 JTextField 中。这里主要用到 JButton、JTextField 组件和面板设置 JPanel 等组件。

【程序实现】

Calculator.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

//计算器显示结果的窗体
class Result extends JPanel {
    //text 先是输入和结果
    JTextField text = new JTextField("0");
    Result() {
        //讲输入的数字或得到的结果在 text 的右边显示
        text.setHorizontalAlignment(SwingConstants.RIGHT);
        text.enable(false);                    //文本框不能编辑
        setLayout(new BorderLayout());         //设定布局管理器边框布局
        add(text, BorderLayout.CENTER);        //text 放置在窗体的中间
    }
}
//计算器数字按钮定义面板
class Number_Key extends JPanel {
    JButton zero = new JButton("0");           //数字键 0
    JButton one = new JButton("1");            //数字键 1
    JButton two = new JButton("2");            //数字键 2
    JButton three = new JButton("3");          //数字键 3
    JButton four = new JButton("4");           //数字键 4
    JButton five = new JButton("5");           //数字键 5
    JButton six = new JButton("6");            //数字键 6
    JButton seven = new JButton("7");          //数字键 7
    JButton eight = new JButton("8");          //数字键 8
    JButton nine = new JButton("9");           //数字键 9
    JButton plus = new JButton("+");
```

```
    JButton sub = new JButton("-");
    JButton mul = new JButton("*");
    JButton div = new JButton("/");
    JButton equal = new JButton("=");
    JButton ce = new JButton("ce");          //置零键
    JButton point = new JButton(".");
    JButton tzero = new JButton("00");
    Number_Key() {
        //定义布局管理器为网格布局
        setLayout(new GridLayout(6, 3, 10, 10));
        //添加各个按钮键
        add(seven);
        add(eight);
        add(nine);
        add(four);
        add(five);
        add(six);
        add(one);
        add(two);
        add(three);
        add(zero);
        add(tzero);
        add(plus);
        add(sub);
        add(mul);
        add(div);
        add(point);
        add(equal);
        add(ce);
    }
}
//计算器主类
public class Calculator extends JFrame implements ActionListener {
    Result result = new Result();                          //定义 text 的面板
    Number_Key number_key = new Number_Key();              //定义按钮面板
    //当单击按钮+、-、*、/时, com = true
    boolean com = false;
    //当 i=0 时说明是我们第一次输入, 字符串 sum 不会累加
    int i = 0;
    //存放 text 的内容
    String sum = "";
    //存放单击按钮+、-、*、/之前的数值
    double total = 0;
    //+、-、*、/的代号分别为 1, 2, 3, 4
    int symbol = 0;
    Calculator() {
        super("Calculator");                               //设定标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);    //设定关闭窗体时退出程序
        JPanel pane = new JPanel();                        //定义主面板
        pane.setLayout(new BorderLayout());
        pane.add(result, BorderLayout.NORTH);
        pane.add(number_key, BorderLayout.SOUTH);
        number_key.one.addActionListener(this);            //对 1 按钮添加监听事件
        number_key.two.addActionListener(this);            //对 2 按钮添加监听事件
        number_key.three.addActionListener(this);          //对 3 按钮添加监听事件
        number_key.four.addActionListener(this);           //对 4 按钮添加监听事件
        number_key.five.addActionListener(this);           //对 5 按钮添加监听事件
        number_key.six.addActionListener(this);            //对 6 按钮添加监听事件
        number_key.seven.addActionListener(this);          //对 7 按钮添加监听事件
        number_key.eight.addActionListener(this);          //对 8 按钮添加监听事件
        number_key.nine.addActionListener(this);           //对 9 按钮添加监听事件
        number_key.zero.addActionListener(this);           //对 0 按钮添加监听事件
        number_key.ce.addActionListener(this);             //对置零按钮添加监听事件
```

```
            number_key.plus.addActionListener(this);          //对+按钮添加监听事件
            number_key.equal.addActionListener(this);         //对=按钮添加监听事件
            number_key.sub.addActionListener(this);           //对-按钮添加监听事件
            number_key.mul.addActionListener(this);           //对*按钮添加监听事件
            number_key.div.addActionListener(this);           //对/按钮添加监听事件
            number_key.tzero.addActionListener(this);         //对 00 按钮添加监听事件
            number_key.point.addActionListener(this);         //对.按钮添加监听事件
            setContentPane(pane);
            pack();//初始化窗体大小为正好盛放所有按钮
        }
        //各个按钮触发的事件
        public void actionPerformed(ActionEvent e) {
            /*
     * 如果是单击数字按钮那么先要判断是否在此之前单击了+、-、*、/、=，如果是，那么 com=true，如果没
有，com=false;或者是否单击数字键，如果是，i = 1,如果没有，i = 0
             */
            if (e.getSource() == number_key.one) {
                if (com || i == 0) {
                    result.text.setText("1");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
                    result.text.setText(sum + "1");
                }
            } else if (e.getSource() == number_key.two) {
                if (com || i == 0) {
                    result.text.setText("2");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
                    result.text.setText(sum + "2");
                }
            } else if (e.getSource() == number_key.three) {
                if (com || i == 0) {
                    result.text.setText("3");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
                    result.text.setText(sum + "3");
                }
            } else if (e.getSource() == number_key.four) {
                if (com || i == 0) {
                    result.text.setText("4");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
                    result.text.setText(sum + "4");
                }
            } else if (e.getSource() == number_key.five) {
                if (com || i == 0) {
                    result.text.setText("5");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
                    result.text.setText(sum + "5");
                }
            } else if (e.getSource() == number_key.six) {
                if (com || i == 0) {
                    result.text.setText("6");
                    com = false;
                    i = 1;
                } else {
                    sum = result.text.getText();
```

```
                result.text.setText(sum + "6");
            }
        } else if (e.getSource() == number_key.seven) {
            if (com || i == 0) {
                result.text.setText("7");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                result.text.setText(sum + "7");
            }
        } else if (e.getSource() == number_key.eight) {
            if (com || i == 0) {
                result.text.setText("8");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                result.text.setText(sum + "8");
            }
        } else if (e.getSource() == number_key.nine) {
            if (com || i == 0) {
                result.text.setText("9");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                result.text.setText(sum + "9");
            }
        }
        /*
     * 对于 0 这个按钮有一定的说法，在此程序中不会出现如 00000 这样的情况，加的判断条件就是
     * 如果 text 中的数值=0 就要判断在这个数值中是否有.存在?如果有那么就在原来数值基础之上添加 0;否则保持原来的数值不变
         */
        else if (e.getSource() == number_key.zero) {
            if (com || i == 0) {
                result.text.setText("0");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                if (Float.parseFloat(sum) > 0 || Float.parseFloat(sum) < 0) {
                    result.text.setText(sum + "0");
                } else {
                    if (sum.trim().indexOf(".") == -1) {
                        result.text.setText(sum);
                    } else {
                        result.text.setText(sum + "0");
                    }
                }
            }
        } else if (e.getSource() == number_key.ce) {
            result.text.setText("0");
            i = 0;
            com = true;
        } else if (e.getSource() == number_key.tzero) {
            if (com || i == 0) {
                result.text.setText("0");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                if (Float.parseFloat(sum) > 0 || Float.parseFloat(sum) < 0) {
                    result.text.setText(sum + "00");
                } else {
                    if (sum.trim().indexOf(".") == -1) {
                        result.text.setText(sum);
                    } else {
                        result.text.setText(sum + "00");
```

```
                    }
                }
            }
        }
        /*
* 不让一个数值中出现 2 个以上的小数点，具体做法是：判断是否已经存在，存在就不添加，不存在就添加
         */
        else if (e.getSource() == number_key.point) {
            if (com || i == 0) {
                result.text.setText("0.");
                com = false;
                i = 1;
            } else {
                sum = result.text.getText();
                if (sum.trim().indexOf(".") == -1) {
                    result.text.setText(sum + ".");
                } else {
                    result.text.setText(sum);
                }
            }
        }//获得单击+之前的数值
        else if (e.getSource() == number_key.plus) {
            com = true;
            i = 0;
            total = Double.parseDouble(result.text.getText());
            symbol = 1;
        }//获得单击-之前的数值
        else if (e.getSource() == number_key.sub) {
            com = true;
            i = 0;
            total = Double.parseDouble(result.text.getText());
            symbol = 2;
        }//获得单击*之前的数值
        else if (e.getSource() == number_key.mul) {
            com = true;
            i = 0;
            total = Double.parseDouble(result.text.getText());
            System.out.println(total);
            symbol = 3;
        }//获得单击/之前的数值
        else if (e.getSource() == number_key.div) {
            com = true;
            i = 0;
            total = Double.parseDouble(result.text.getText());
            symbol = 4;
        } else if (e.getSource() == number_key.equal) {
            switch (symbol) {
            case 1://计算加法
            {
                double ad = total + Double.parseDouble(result.text.getText());
                result.text.setText(ad + "");
                i = 0;
                sum = "";
                break;
            }
            case 2://计算减法
            {
                double ad = total - Double.parseDouble(result.text.getText());
                result.text.setText(String.valueOf(ad));
                i = 0;
                sum = "";
                break;
            }
            case 3://计算乘法
            {
                double ad = total * Double.parseDouble(result.text.getText());
                result.text.setText(ad + "");
                i = 0;
```

```
                    sum = "";
                    break;
                }
                case 4://计算除法
                {
                    double ad = total / Double.parseDouble(result.text.getText());
                    result.text.setText(ad + "");
                    i = 0;
                    sum = "";
                    break;
                }
                }
                System.out.println(com);
            }
        }
        public static void main(String[] args) {
            Calculator calculator = new Calculator();
            calculator.show();
        }
}
```

【运行结果】

运行结果如图 5-14 所示。



图 5-14

【分析与总结】

在版面布局上定义了两个面板，一个是用来放置和布局所有按钮，用的版面控制器是 GridLayout，设为 6 行 3 列，并设置了组件与组件之间的距离为 10；另一个用来放置 JTextField。

在事件处理中，actionPerformed()是这个程序最重要的函数，每个按钮都与数字、运算符或者功能符一一对应，通过 ActionEvent 的 getSource()得到被击发的按钮，从而获得输入，然后通过两个标记 i 和 com 记忆上一次输入的情况，当前输入的是数字，如果上一次输入的是数字，那么就将这次输入的数字加到原来 text 的末尾，如果上一次输入的是算术符，那么就重置 text；当前输入的是算术符，则计算当前结果，total = total 运算符 text。最后再加上一些特殊情况的处理，比如消除 0000 的情况出现等等。

实例 63 TicTacToe 游戏

本实例实现简单的 TicTacToe 小游戏。

【编程要点】

每个方格都由 JPanel 构成，通过捕获鼠标事件来获得用户的输入，并在 JPanel 上画 X 和画 O 来分游戏的两方，当 X 方或者 O 方占满某行、某列或对角线时，该方获胜。为了程序的扩展性，可以由用户输入行数和列数来生成游戏方格，通过 JDialog 来实现。

【程序实现】

```
//: c14:TicTacToe.java

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TicTacToe extends JApplet {
    private JTextField rows = new JTextField("3"),
                       cols = new JTextField("3");
    private static final int BLANK = 0, XX = 1, OO = 2;
    class ToeDialog extends JDialog {
        private int turn = XX; //从 X 开始
        ToeDialog(int cellsWide, int cellsHigh) {
            setTitle("The game itself");
            Container cp = getContentPane();
            cp.setLayout(new GridLayout(cellsWide, cellsHigh));
            for (int i = 0; i < cellsWide * cellsHigh; i++)
                cp.add(new ToeButton());
            setSize(cellsWide * 50, cellsHigh * 50);
            setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        }
        class ToeButton extends JPanel {
            private int state = BLANK;
            public ToeButton() {
                addMouseListener(new ML());
            }
            public void paintComponent(Graphics g) {
                super.paintComponent(g);
                int x1 = 0, y1 = 0;
                int x2 = getSize().width - 1;
                int y2 = getSize().height - 1;
                g.drawRect(x1, y1, x2, y2);
                x1 = x2 / 4;
                y1 = y2 / 4;
                int wide = x2 / 2, high = y2 / 2;
                if (state == XX) {
                    g.drawLine(x1, y1, x1 + wide, y1 + high);
                    g.drawLine(x1, y1 + high, x1 + wide, y1);
                }
                if (state == OO)
                    g.drawOval(x1, y1, x1 + wide / 2, y1 + high / 2);
            }
            class ML extends MouseAdapter {
                public void mousePressed(MouseEvent e) {
                    if (state == BLANK) {
                        state = turn;
                        turn = (turn == XX ? OO : XX);
                    } else
                        state = (state == XX ? OO : XX);
                    repaint();
                }
            }
        }
    }
    class BL implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            JDialog d = new ToeDialog(Integer.parseInt(rows.getText()), Integer
                    .parseInt(cols.getText()));
            d.setVisible(true);
        }
    }
    public void init() {
        JPanel p = new JPanel();
        p.setLayout(new GridLayout(2, 2));
        p.add(new JLabel("Rows", JLabel.CENTER));
        p.add(rows);
        p.add(new JLabel("Columns", JLabel.CENTER));
        p.add(cols);
        Container cp = getContentPane();
```

```
        cp.add(p, BorderLayout.NORTH);
        JButton b = new JButton("go");
        b.addActionListener(new BL());
        cp.add(b, BorderLayout.SOUTH);
    }
    public static void main(String[] args) {
        TicTacToe test = new TicTacToe();
        JFrame frame = new JFrame("TicTacToe");
        frame.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        frame.getContentPane().add(test);
        frame.setSize(100,100);
       test.init();
       test.start();
       frame.setVisible(true);
    }
}
```

【运行结果】

运行结果如图 5-15 所示。

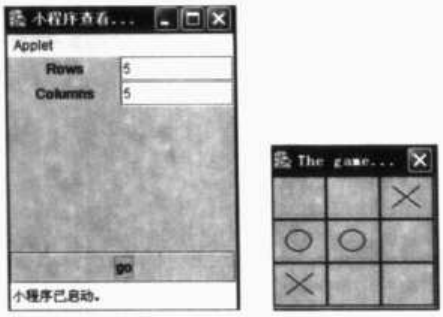


图 5-15

【分析与总结】

这里主要演示的是小游戏的界面设计，因此没有加入判别游戏双方的胜负方法，主要是 paintComponent()函数中，在当前的 JPanel 内调用画线函数或画椭圆的函数来显示两方的输入情况。

实例 64 文本编辑器

本实例创建一个简单的文本编辑器，包括文件新建、保存、另存为、打开和剪切、复制、粘贴、全选等功能。

【编程要点】

文件的打开、保存等用到了 FileDialog 类：

FileDialog(this, "Open File",FileDialog.LOAD)：打开模式。

FileDialog(this,"Save File",FileDialog.SAVE)：保存模式。

getDirectory()：获取所选文件的路径。

getFile()：获取所选文件的文件名。

有关 TextArea 的方法：

selectAll()：全选。

getSelectedText()：获取所选的字符串。

getSelectionStart()：获取所选字符串的终点位置。

replaceRange(String s, int start, int end)：用 String 指定的内容替换 start 和 end 之间的字符子串。

有关剪贴板的类：

Clipboard 类：用 Clipboard clipboard = toolkit.getSystemClipboard()获取系统剪贴板。

Transferable 接口：可传输对象接口，只有实现该接口的对象才能放入剪贴板中。

StringSelection 类：实现 Transferable 接口，可放入剪贴板中，用来传输文本数据。

DataFlavor 类：分析剪贴板中的数据类型，如 DataFlavor.stringFlavor 表示 Unicode 的字符数据，DatafFlavor.imageFlavor 表示图形数据。

【程序实现】

```
//NotePad.java
import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class NotePad extends JFrame implements ActionListener {
    JTextArea textarea = new JTextArea();
    JMenuBar menubar = new JMenuBar();
    JMenu filemenu = new JMenu("File");
    JMenuItem newitem = new JMenuItem("New");
    JMenuItem openitem = new JMenuItem("Open");
    JMenuItem saveitem = new JMenuItem("Save");
    JMenuItem saveasitem = new JMenuItem("Save As");
    JMenuItem exititem = new JMenuItem("Exit");
    JMenu editmenu = new JMenu("Edit");
    JMenuItem selectitem = new JMenuItem("Select All");
    JMenuItem copyitem = new JMenuItem("Copy");
    JMenuItem cutitem = new JMenuItem("Cut");
    JMenuItem pasteitem = new JMenuItem("Paste");
    JMenu aboutmenu = new JMenu("About");
    JMenuItem info = new JMenuItem("About NotePad");
    String fileName = "NoName";  //设置默认的文件名
    Toolkit toolkit = Toolkit.getDefaultToolkit();
    Clipboard clipboard = toolkit.getSystemClipboard();  //剪贴板对象
    private FileDialog openFileDialog = new FileDialog(this, "Open File",
            FileDialog.LOAD);
    private FileDialog saveAsFileDialog = new FileDialog(this, "Save File As",
            FileDialog.SAVE);
    public NotePad() {
        setTitle("NotePad");
        setFont(new Font("Times New Roman", Font.PLAIN, 12));
        setBackground(Color.white);
        setSize(400, 300);
        filemenu.add(newitem);
        filemenu.add(openitem);
        filemenu.addSeparator();
        filemenu.add(saveitem);
        filemenu.add(saveasitem);
        filemenu.addSeparator();
        filemenu.add(exititem);
        editmenu.add(selectitem);
        editmenu.add(copyitem);
        editmenu.addSeparator();
        editmenu.add(cutitem);
        editmenu.add(pasteitem);
        menubar.add(filemenu);
        menubar.add(editmenu);
        menubar.add(aboutmenu);
        aboutmenu.add(info);
        setJMenuBar(menubar);
        getContentPane().add(textarea);
        addWindowListener(new WindowAdapter() {
```

```
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        newitem.addActionListener(this);
        openitem.addActionListener(this);
        saveitem.addActionListener(this);
        saveasitem.addActionListener(this);
        exititem.addActionListener(this);
        selectitem.addActionListener(this);
        copyitem.addActionListener(this);
        cutitem.addActionListener(this);
        pasteitem.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        Object eventSource = e.getSource();//利用事件源判断被单击的菜单项
        if (eventSource == newitem) {
            textarea.setText("");
        } else if (eventSource == openitem) {
            openFileDialog.show();
            fileName = openFileDialog.getDirectory() + openFileDialog.getFile();
            if (fileName != null)
                readFile(fileName);
        } else if (eventSource == saveitem) {
            if (fileName != null)
                writeFile(fileName);
        } else if (eventSource == saveasitem) {
            saveAsFileDialog.show();
            fileName = saveAsFileDialog.getDirectory()
                    + saveAsFileDialog.getFile();
            if (fileName != null)
                writeFile(fileName);
        } else if (eventSource == selectitem) {
            textarea.selectAll();
        } else if (eventSource == cutitem) {
            String text = textarea.getSelectedText();
            StringSelection selection = new StringSelection(text);
            clipboard.setContents(selection, null);
            textarea.replaceRange("", textarea.getSelectionStart(), textarea
                    .getSelectionEnd());
        } else if (eventSource == pasteitem) {
            Transferable contents = clipboard.getContents(this);
            if (contents == null)
                return;
            String text;
            text = "";
            try {
                text = (String) contents
                        .getTransferData(DataFlavor.stringFlavor);
            } catch (Exception exception) {
            }
            textarea.replaceRange(text, textarea.getSelectionStart(), textarea
                    .getSelectionEnd());
        } else if (eventSource == exititem) {
            System.exit(0);
        }
    }
    //读文件
    public void readFile(String fileName) {
        try {
            File file = new File(fileName);
            FileReader readIn = new FileReader(file);
            int size = (int) file.length();
            int charsRead = 0;
            char[] content = new char[size];
            while (readIn.ready())
                charsRead += readIn.read(content, charsRead, size - charsRead);
            readIn.close();
            textarea.setText(new String(content, 0, charsRead));
        } catch (IOException e) {
```

```
                System.out.println("Error Opening file");
            }
        }
        //写文件
        public void writeFile(String fileName) {
            try {
                File file = new File(fileName);
                FileWriter writeOut = new FileWriter(file);
                writeOut.write(textarea.getText());
                writeOut.close();
            } catch (IOException e) {
                System.out.println("Error writing file");
            }
        }
        public static void main(String[] args) {
            Frame frame = new NotePad();
            frame.show();
        }
}
```

【运行结果】

运行结果如图 5-16 所示。



图 5-16

【分析与总结】

程序中使用 Clipboard.setContent(Transferable contents, ClipboardOwner owner)把数据放入剪贴板，其中 content 可以是实现了 Transferable 的任何类。这里是 StringSelection 类。然后使用 Transferable Clipboard.getContent(Object requestor)读取剪贴板中的数据，其中 requestor 是要获取数据的对象。本例是一个 EditorFrame 对象，所以用 this 代替，该函数返回一个 Transferable 接口。另外，本程序还可以加入更多的组件和事件处理来作一个更完善的记事本。

小结

Swing 组件是 Sun 公司推出来用于替代 AWT 的轻量组件。Swing 组件除提供了与 AWT 组件兼容的功能外，还提供了许多增强的功能。如 Swing 标签组件（JLabel）和按钮组件（JButton）可以显示图标，而 AWT 则不能。本章重点掌握各种 Swing 的组件，如 JTabel、JTree、Jlist 等 Swing 体系结构中最基础、最常用的组件。

第 6 章　Java 多媒体与图形图像处理编程

本章通过 11 个实例介绍了图形、图像和音频处理，包括画图、填充图形、动态填充、字体、数字时钟、纹理与色彩、模拟时钟、椭圆形按钮、裁剪效果、Applet 音频播放器、音频播放等。

实例 65　画图

本实例演示 Java 2D 的画图方法。

【编程要点】

Java 2D API 是 JFC(Java Fundation Classes)的一员，表 6-1 与表 6-2 列出了与 Graphics2D 上下文一起使用的接口和类，包括代表状态属性的类。这些类中的大多数都是 java.awt 包的组成部分。

表 6-1　与 Graphics2D 上下文一起使用的接口

接口	说明
Composite	定义基本图形元素和底层图形区域的复合方法。由 AlphaComposite 实现
CompositeContext	定义复合操作的封装和优化环境。用于程序员实现自定义复合规则
Paint	扩展：Transparency 定义 draw 或 fill 操作的颜色。由 Color、GradientPaint 和 TexturePaint 实现
PaintContext	定义绘画操作的封装和优化环境。程序员用来实现自定义绘画操作
Stroke	生成 Shape 并包含要绘制的 Shape 的轮廓。由 BasicStroke 实现

表 6-2　与 Graphics2D 上下文一起使用的类

类	说明
AffineTransform (java.awt.geom)	代表 2D 仿射变换，用于执行从一种二维坐标到其他二维坐标的线性映射
AlphaComposite	实现：Composite 实现形状、文本和图像的基本 alpha 复合规则
BasicStroke	实现：Stroke 定义应用于 Shape 轮廓的“画笔样式”
Color	实现：Paint 定义 Shape 的实心填充颜色
GradientPaint	实现：Paint 定义 Shape 的线性颜色渐变填充图案。该填充图案将从点 P1 的 C1 颜色逐渐变为点 P1 的 C2 颜色
Graphics2D	扩展：Graphics 2D 绘制的基本类。是原来的 java.awt.Graphics 类的扩展
TexturePaint	实现：Paint 定义 Shape 的纹理或图案填充。纹理或图案由 BufferedImage 生成

Graphics2D 加强了传统 AWT（Abstract Windowing Toolkit）的描绘功能。透过 Java 2D API，Programmer 可以轻松地描绘出任意的几何图形、运用不同的填色效果、对图形做旋转（rotate）、缩放（scale）、扭曲（shear）等。

java.awt 包含了一些新增的 2D API 类别（classes）和界面（interfaces）。其中 Graphics2D 继承自 java.awt.Graphics，是描绘 2D 图形的物件（object）。如同前版本的 JDK 所使用的绘图模式一样，当有物件要被描绘时，paint 或是 update method 会自动根据适当的 Graphics Context 来做绘图的工作。所谓的 Graphics Context 是与 Graphics Object 相关状态属性（state attribute）的集合。在 Graphics2D 中新增了许多状态属性，像 Stroke、Paint、Clip、Transform 等。

java.awt.geom 则包含可以勾勒任何形状的 GeneralPath 类别。它可以由许多不同种类的 subpath 构成，像 lines 和 quadratic curves 等。为了兼顾方便性，此 package 中更定义了许多基本几何图形，包括 Arc2D、CubicCurve2D、Line2D 等。这些类别都有两种形态，分别是 Float precision 与 Double precision。此外，还有一种特别的图形类别：Area，它支持 Constructive Area Geometry（CGA）。CGA 是对既有图形物件做布尔运算（Boolean Operation）而产生新图形物件的过程。

java.awt.Shape 接口是一个代理，它描述几何形状的轮廓和特征。下面是 java.awt.geom 中的几个基本的画图的类的继承关系，如图 6-1 所示。

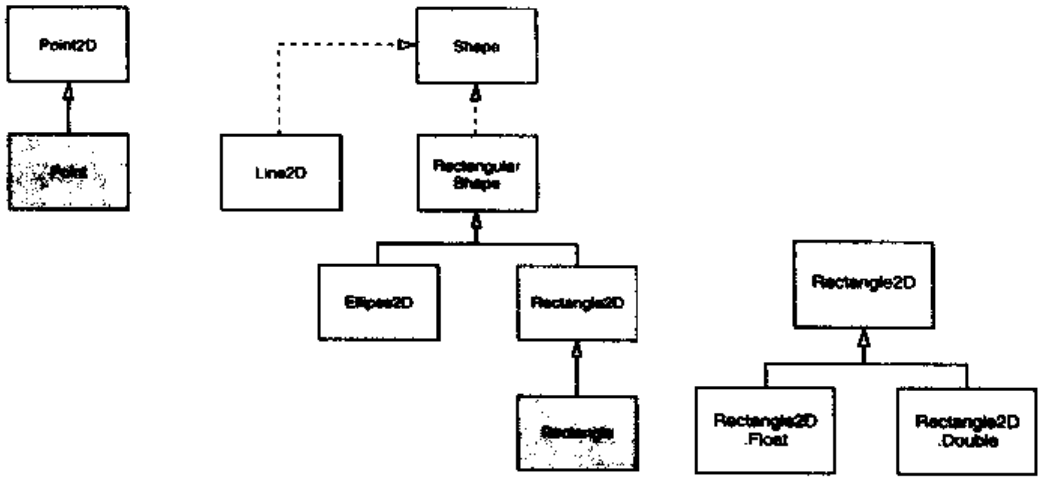


图 6-1

下面是几个类的构造函数：

Line2D.Double(double X1, double Y1, double X2, double Y2)。其中 X1、Y1 为线段起点的坐标，X2、Y2 为线段终点的坐标。

Rectangle2D.Double(double X1, double Y1, double w, double h)。其中 X1、Y1 是矩形左上角顶点的坐标，w、h 分别为矩阵的宽和高。

Ellipse2D.Double(double x, double y, double w, double h)。其中，x、y 为椭圆的左上角顶点的坐标，w、h 分别是椭圆的宽和高。

Ellipse2D.Double 中的 setFrame(Rectangle2D r)方法，是用矩形 r 的参数赋给椭圆，它们之间存在一一对应关系。

Ellipse2D.Double 中的 setFrameFromCenter(double centerX, double centerY, double

cornerX, double cornerY)。其中 centerX、centerY 表示椭圆的中心坐标，cornerX、cornerY 表示椭圆的右上角顶点的坐标。注意，这里所指的椭圆的顶点都是指椭圆的外接矩形的顶点。

画圆形是通过画椭圆的类来实现，只需将椭圆的宽和高设为相同即可。

【程序实现】

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class DrawDemo {
    public static void main(String[] args) {
        DrawFrame frame = new DrawFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();
    }
}
/**
  一个 frame 中包含一个 panel 用来画图
 */
class DrawFrame extends JFrame {
    public DrawFrame() {
        setTitle("DrawDemo");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        //将 panel 加到 frame
        DrawPanel panel = new DrawPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }

    public static final int DEFAULT_WIDTH = 400;
    public static final int DEFAULT_HEIGHT = 400;
}

/**
 用来显示矩形和椭圆的 panel
 */
class DrawPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        //画矩形
        double leftX = 100;
        double topY = 100;
        double width = 200;
        double height = 150;

        Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
        g2.draw(rect);

        //画 rect 的内切椭圆
        Ellipse2D ellipse = new Ellipse2D.Double();
        ellipse.setFrame(rect);
        g2.draw(ellipse);

        //画一条对角线
        g2.draw(new Line2D.Double(leftX, topY, leftX + width, topY + height));

        //画一个同心的圆
        double centerX = rect.getCenterX();
        double centerY = rect.getCenterY();
        double radius = 150;

        Ellipse2D circle = new Ellipse2D.Double();
```

```
            circle.setFrameFromCenter(centerX, centerY, centerX + radius, centerY
                    + radius);
            g2.draw(circle);
        }
    }
```

【运行结果】

运行结果如图 6-2 所示。

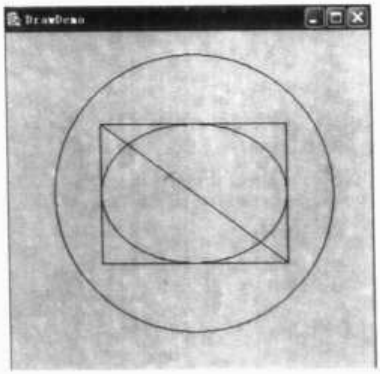


图 6-2

【分析与总结】

这个程序的基本结构是继承一个 JPanel 类，将图形画在该自定义的 Panel 上，然后将该 Panel 加到主 Frame 中。

最重要的部分是 DrawPanel 类中的 paintComponent 方法，所有的画图工作都在此完成。

实例 66　填充图形

本实例演示如何在图形中填充指定颜色。

【编程要点】

Graphics2D 类描述了两种特性，Stroke 和 Paint，它们含有一些代理对象，这些对象描述了用来绘制和填充形状的虚拟笔和刷子。Paint 接口有几个具体的实现，它们允许用纯色、渐变色或图案来填充形状。

对 java.awt.Color 类做了一些调整以实现 Paint，并且可以用于纯色填充。java.awt.GradientPaint 类允许用线性颜色渐变色来填充形状，线性颜色渐变色允许在两个指定的 Color 对象之间创建过渡。可以将渐变色设置成“周期性的”，这将导致渐变色图案重复出现。最后，还提供了 java.awt.TexturePaint 类，它用由 BufferedImage 描述的图案填充形状。

【程序实现】

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class FillDemo {
    public static void main(String[] args) {
        FillFrame frame = new FillFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.show();
    }
}
```

```
/**
 包含一个用于 panel 的 frame
 */
class FillFrame extends JFrame {
    public FillFrame() {
        setTitle("FillTest");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        //将 panel 加到 frame 中
        FillPanel panel = new FillPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }

    public static final int DEFAULT_WIDTH = 400;
    public static final int DEFAULT_HEIGHT = 400;
}

/**
 用于显示填充矩形和椭圆的 panel
 */
class FillPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;

        //画矩形
        double leftX = 100;
        double topY = 100;
        double width = 200;
        double height = 150;

        Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);
        g2.setPaint(Color.RED);
        g2.fill(rect);

        //画椭圆
        Ellipse2D ellipse = new Ellipse2D.Double();
        ellipse.setFrame(rect);
        g2.setPaint(Color.WHITE);
        g2.fill(ellipse);
    }
}
```

【运行结果】

运行结果如图 6-3 所示。



图 6-3

【分析与总结】

本实例与上一例的基本内容相同，Graphics2D 类中的 setPaint()的方法设置画笔的颜色，然后通过 fill(Shape s)将颜色填充到图形 s 中。

实例 67 动态填充

本实例通过多线程来实现动态填充。

【编程要点】

利用多线程的技术，通过实现 Runnable 接口来实现。覆写 run 方法，每隔一段时间调用重画操作，从而达到动态效果。

【程序实现】

```
import java.awt.*;
import javax.swing.*;

public class BouncingCircle extends JApplet implements Runnable {
    int x = 150, y = 50, r = 50;    //圆的位置和半径
    int dx = 11, dy = 7;            //圆的轨迹
    Thread animator;                //产生动态效果的线程
    volatile boolean pleaseStop;    //线程结束的标记
    //在当前的位置画圆
    public void paint(Graphics g) {
        g.setColor(Color.red);
        g.fillOval(x - r, y - r, r * 2, r * 2);
    }

    /**
     * 重新设定圆的位置，并执行重画操作
     * 不断被线程调用
     **/

    public void animate() {
        //到达边界时反弹
        Rectangle bounds = getBounds();
        if ((x - r + dx < 0) || (x + r + dx > bounds.width))
            dx = -dx;
        if ((y - r + dy < 0) || (y + r + dy > bounds.height))
            dy = -dy;

        //重新设定圆的位置
        x += dx;
        y += dy;

        repaint();
    }

    //实现 Runnable 接口的 run 方法
    public void run() {
        while (!pleaseStop) { //Loop until we're asked to stop
            animate();                  //重画操作
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {
            }
        }
    }

    /** 当 applet 被加载时启动线程 */
    public void start() {
        animator = new Thread(this);    //创建线程
        pleaseStop = false;
        animator.start();
    }

    /** 当 applet 关闭时线程停止 */
    public void stop() {
```

```
            pleaseStop = true;
        }
    }
```

【运行结果】

运行结果如图 6-4 所示。



图 6-4

【分析与总结】

每一次重画的圆的位置都不一样，在 animate()方法中不断地调整画圆的位置，由于画圆的参数 x、y 是递减的，当到达边界时，再反弹，所以有些位置无法到达，因此不能将整个 Applet 窗口填满。

实例 68　字体

本实例演示字体的处理。

【编程要点】

Font 产生字体对象的类。构造函数为 Font（String name, int style, int size）使用指定的名称，风格和字号来创建一个新的字体。其中 name 为字体的名称，style 为被使用的风格常量，size 为字体的字号。Font 提供的方法 getStringBounds(String message, FontRenderContext fre)获得字符串的边界，返回一个 Rectangle2D 类型的对象。

字体渲染是一种高度复杂的机制，因为字体是由许多复杂的形状组成的，而这些形状被应用了很多几何变换。为了将渲染能力同图形环境以及字体渲染引擎分隔开来，使用了一个名为 java.awt.font.FontRenderContext 的类。该类封装了要正确地确定文本的可查看大小所需的信息。

【程序实现】

```
import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.*;

public class FontDemo
{
   public static void main(String[] args)
   {
      FontFrame frame = new FontFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

class FontFrame extends JFrame
{
   public FontFrame()
   {
```

```
        setTitle("FontDemo");
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
        FontPanel panel = new FontPanel();
        Container contentPane = getContentPane();
        contentPane.add(panel);
    }

    public static final int DEFAULT_WIDTH = 300;
    public static final int DEFAULT_HEIGHT = 200;
}

class FontPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        String message = "Hello, World!";
        //创建字体对象
        Font f = new Font("Serif", Font.BOLD, 36);
        g2.setFont(f);

        //获得文本字体的大小
        FontRenderContext context = g2.getFontRenderContext();
        Rectangle2D bounds = f.getStringBounds(message, context);

        //得到文本的左上角坐标(x,y)
        double x = (getWidth() - bounds.getWidth()) / 2;
        double y = (getHeight() - bounds.getHeight()) / 2;

        //计算下划线的 y 坐标值
        double ascent = -bounds.getY();
        double baseY = y + ascent;

        //显示文本信息
        g2.drawString(message, (int)x, (int)(baseY));
        g2.setPaint(Color.BLUE);

        //画下划线
        g2.draw(new Line2D.Double(x, baseY,
            x + bounds.getWidth(), baseY));

        //画包围框
        Rectangle2D rect = new Rectangle2D.Double(x, y,
            bounds.getWidth(),
            bounds.getHeight());
        g2.draw(rect);
    }
}
```

【运行结果】

运行结果如图 6-5 所示。



图 6-5

【分析与总结】

Graphics 类中的 setFont(Font)的将指定的字体设置为该图形上下文的字体。随后所有使用该图形上下文的文本操作都使用该字体；getFontRenderContext()方法返回文本字体的

上下文信息。

实例69 数字时钟

本实例演示字体处理。

【编程要点】

虽然Graphics2D中仍然提供 drawString()的旧版本(该版本接收一个String作为参数),但Graphics2D中已经有了更为强大的能力，允许渲染格式化字符串。

java.text.AttributedString 类允许构造含有属性信息的字符串。当使用 AttributedString 时，它总是由实现了 java.text.AttributedCharacterIterator 接口的对象来指定。其迭代器类似于 GeneralPath 的迭代器，因为它允许文本布局和渲染类将文本分割成一个个属性序列。要创建 AttributedString，只须使用 String 对象或现有的 AttributedString。使用 addAttribute() 方法创建了 AttributedString 之后，就可以为其提供属性。该方法获取一个 AttributedCharacterIterator.Attribute 类型的参数，以及一个可选的属性序列的起始和终止索引。省略这些索引将导致将指定的属性应用于整个字符串。

可以用于 AttributedString 的部分属性有：

BACKGROUND：控制背景颜色；可以设置为透明。

FOREGROUND：控制前景颜色；覆盖图形环境的 Paint 特性。

FONT：控制用于渲染字符的字体；FONT 覆盖图形环境的 Font 特性。

FAMILY、SIZE 和 WEIGHT：提供另一种方法来指定字符字体，而不是直接创建 Font 对象。这些属性也代替了图形环境的 Font 特性。

CHAR_REPLACEMENT：允许用字符替换任意 Shape 对象。这是另外一种创建定制字体的方法。

TRANSFORM：对 AttributedString 中的全部或部分字符应用某个变换。

java.awt.font.TextLayout 类提供带属性的字符的图形表示。TextLayout 用于从 AttributedString 派生诸如被渲染的文本将有多大之类的信息；它还包含一些方法，提供插入记号(表示文本中的插入点的闪烁形状)、突出显示和单击测试。注：TextLayout 同控制容器中组件布局的 java.awt.LayoutManager 类无关。TextLayout 是一种不可变对象，这意味着一旦被构建，其任何特性都无法被更改。如果源 AttributedString 改变了，那么就必须创建一个新 TextLayout 实例。从处理器和内存开销的意义上说，它是一种成本相对较高的对象，因此使用它的程序必须尽可能少地创建 TextLayout 实例。Graphics2D 并没有任何从 TextLayout 进行绘制的方法，但 TextLayout 却有它自己的 draw() 方法版本，它的这一方法获取一个 Graphics2D 作为其参数之一。

TextLayout 中最常用的信息元素是一些只读特性，它们返回有关显示渲染的字符串所需大小的信息。所有这些特性都是 float 类型：

getAdvance()：渲染字符串的宽度。

getAscent()：渲染字符串从布局顶部到基线(一条想象中的、沿着字符底部的线)的高度。

getDescent()：渲染字符串从基线到布局底部的高度。它是“p”和“q”之类的小写字

符从那条线下面扩展的区域。

getLeading()：建议的文本行之间的空间。这一术语来源于铅字时代，当时使用一条铅带将打印行分隔开来。

另外由于数字时钟每一秒要更新一次，使用 Timer 组件又可以让程序员在一段时间内依次完成指定的操作，这在动画的展示上非常有用。在 Java 中，Swing 的 Timer 组件就可以让用户做到这样的功能，而且非常容易使用，下面先来看 Timer 的构造函数。

Timer 构造函数：

Timer(int delay,ActionListener listener)：建立一个 Timer 组件，并在每一次 delay 的时间点上触发 ActionEvent。使用 Timer 组件，它会在程序员给予的 delay 时间，周期性的触发 ActionEvent 事件，如果要处理这个事件，必须实现 ActionListener 界面所定义的 actionPerformed()方法。

要开始激活 Timer 组件，可以用 start()方法，要停止 Timer 组件可以使用 stop()方法，要从新激活 Timer 组件可以使用 restart()方法，若只想 Timer 组件只触发一次 ActionEvent 事件，可以利用 setRepeats(false)方法，将参数设为 false，若要设置 delay 时间则可用 setDelay()方法。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.util.*;
import java.text.*;
import javax.swing.*;
import javax.swing.plaf.*;

public class DigitalClock extends JPanel
{
    public DigitalClock()
    {
        //指定一个 UI 对象
        this.setUI(DigitalClockPanelUI.createUI(this));
        //设本 panel 为不透明
        this.setOpaque(true);
    }

    public static void main(String [] args)
    {
        JFrame frame = new JFrame();
        frame.setLocation(200, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DigitalClock panel = new DigitalClock();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}

//继承 PanelUI 类，该类在所有平台上都能使用
class DigitalClockPanelUI extends PanelUI implements ActionListener
{
    protected final static Font clockFont =
                        new Font("Arial", Font.BOLD, 24);
    //时间显示的格式
    protected final static SimpleDateFormat dateFormat =
                        new SimpleDateFormat("hh:mm:ss a");
    protected final static FontRenderContext frc =
                        new FontRenderContext(null, true, true);
```

```
    //字体之间空格
    protected final static int FUDGE = 6;
    //含当前时间的 AttributedString
    protected AttributedString timeString = null;
    protected TextLayout textLayout = null;
    //每秒一次的 timer 控件
    protected javax.swing.Timer timer =
                    new javax.swing.Timer(1000, this);
    protected DigitalClock panel;

    private DigitalClockPanelUI(DigitalClock panel)
    {
        this.panel = panel;
        panel.setBackground(Color.black);
        actionPerformed(null);
        timer.start();
    }

    public static ComponentUI createUI(JComponent component)
    {
        return new DigitalClockPanelUI((DigitalClock) component);
    }

    public void paint(Graphics g, JComponent c)
    {
        //没有时间或是没有 textLayout 时，不进行重画
        if (this.timeString == null || this.textLayout == null)
            return;

        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
        //在 textLayout 指示的位置上显示字符串
        g2.drawString(timeString.getIterator(), 1,
                (int) (textLayout.getAscent()));
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_DEFAULT);
    }

    //当间隔时间到时，timer 会自动调用此方法
    public void actionPerformed(ActionEvent event)
    {
        //为新的时间字符串创建 AttributedString
        timeString = new AttributedString(dateFormat.format(new Date()));
        //设定字体和颜色
        timeString.addAttribute(TextAttribute.FONT, clockFont);
        timeString.addAttribute(TextAttribute.FOREGROUND, Color.red);
        //将中间的时间分隔符设定为黄色
        timeString.addAttribute(TextAttribute.FOREGROUND,
                                Color.yellow, 2, 3);
        timeString.addAttribute(TextAttribute.FOREGROUND,
                                Color.yellow, 5, 6);
        //创建新的 TextLayout
        textLayout = new TextLayout(timeString.getIterator(), frc);
        panel.repaint();
    }
    //返回组件的合适大小
    public Dimension getPreferredSize(JComponent c)
    {
        Dimension size = textLayout.getBounds().getBounds().getSize();
        //适当调大高度和宽度
        size.height += FUDGE;
        size.width += (FUDGE + 2);
        return size;
    }
}
```

【运行结果】

运行结果如图 6-6 所示。



图 6-6

【分析与总结】

java.awt.RenderingHints 类定义了很多方法以显示二维图形，包括 alpha_interpolation、抖动以及 antialiasing 方法。RenderingHints 有助于决定图形如何显示以及图形如何达到最佳处理。Antialiasing 是一种图形的平滑处理方法。其算法是选择一个特殊像素的颜色值并取代交叉处的像素，从而能够使线条交叉处得到平滑化。setRenderingHint() 方法的参数是一个键以及对应的键值。可以看出使用 antialiasing 方法的饼状图形的线条交叉处变得很平滑。

实例 70　纹理与色彩

纹理是指通过在给定区域内排列填充图像块形成的图像效果，本实例实现了这种效果。

【编程要点】

Graphics2D 类允许使用图像块填充某个区域的内部，步骤如下：

（1）创建一个 BufferedImage 类的实例并初始化宽和高。

（2）获取 BufferedImage 的图像。

（3）使用刚刚获得的图像在 BufferedImage 的实例上绘制。

（4）创建一个 TexturePaint 实例指向上述的 buffered image。

（5）使用上面的纹理作为画笔。

（6）画图并填充。

【程序实现】

```
import java.awt.*;
import java.awt.geom.*;
import java.net.*;
import java.awt.image.BufferedImage;
import javax.swing.*;

public class TextureDemo extends JApplet {
  private BufferedImage bImage;

  public void init(){

    //获得图片资源
    URL url=null;
    try {
      url = new URL(getCodeBase(),"images\\Winter.jpg");
    } catch(MalformedURLException e) {
        String msg = "Error loading image duke.gif";
        System.err.println(msg);
        showStatus(msg);
        System.exit(0);
    }

    Image img = new ImageIcon(url).getImage();

    //建立一个和 image 大小相同的 BufferedImage
    //把图片放在上面
```

```
        bImage = new BufferedImage(img.getWidth(null),img.getHeight(null),
                    BufferedImage.TYPE_INT_RGB);
        Graphics2D g2 = bImage.createGraphics();
        g2.drawImage(img,null,null);
    }

    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        //创建一个和图片大小一样的矩形
        Rectangle2D tr = new Rectangle2D.Double(0, 0,
                        bImage.getWidth(), bImage.getHeight());
        //创建纹理画笔
        TexturePaint tp = new TexturePaint(bImage, tr);
        //创建圆角矩形
        RoundRectangle2D r = new RoundRectangle2D.Float
                                (25, 35, 150, 150, 25, 25);
        //用纹理填充矩形
        g2.setPaint(tp);
        g2.fill(r);
        //创建一个椭圆并用纹理填充
        Ellipse2D e = new Ellipse2D.Float(200, 35, 100, 150);
        g2.fill(e);
        //创建一个矩形并设置为thick stroke 使得当前纹理可视
        Rectangle2D r2 = new Rectangle2D.Double(320, 35, 150, 150);
        g2.setStroke(new BasicStroke(20));
        g2.draw(r2);
    }
}
```

【运行结果】

运行结果如图 6-7 所示。



图 6-7

【分析与总结】

要创建自己的图像，最灵活的方法是用一个 BufferedImage 对象，它是 Image 类的一个子类，它把图像数据存储在一个可以被访问的缓冲区中。它还支持各种存储像素数据的方法：使用或不使用 alpha 通道、不同种类的颜色模型以及颜色组件的各种精确度。为了应用这个图像，需要有图形上下文，而 BufferedImage 对象的 createGraphics()方法就返回一个与该图像相关 Graphics 2D 对象：Graphics2D g2D = image.createGraphics()；使用 g2D 对象的操作会修改 BufferedImage 对象 image 的像素。利用这个对象，就完全有能力应用 BufferedImage 对象。可以绘制形状、图像、GeneralPath 对象或任何别的东西，还可以为图形上下文设置 alpha 组合对象。同时还拥有 Graphics2D 对象提供的变形能力。

实例 71 模拟时钟

本实例通过 Java 2D 来创建一个模拟时钟。

【编程要点】

GeneralPath：它是一个 Shape 对象，该对象提供将片段连接成复杂几何形状的命令。它允许使用直线和贝塞尔曲线序列来构建图形，还允许用子路径来组成路径的片段。常用的两种方法 lineTo(float x, float y)、moveTo(float x, float y)，前者表示从当前点连直线到点（x，y），后者表示从当前点移动到点（x，y）。

Graphics2D 还提供多种方法改变 transform 属性，可以构建一个新的 AffineTransform 并通过调用 transform 改变该属性。

AffineTransform 定义了 getRotateInstance、getScaleInstance、getShearInstance 和 getTranslateInstance 方法来简化 new transform 的构建。也可一使用某个 Graphics2D transformation 方法改变当前 transform。调用这类方法时，显示的变换结果和如下当前变换紧密关联。

（1）rotate：指定以弧度为单位的旋转角度。

（2）scale：分 x 和 y 方向指定倾斜度。

（3）shear：分 x 和 y 方向指定斜拉度。

（4）translate：分 x 和 y 方向指定 translation offset。

还可直接构建一个 AffineTransform 并通过调用 transform 方法将其与当前的 transform 关联。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.Calendar;
import javax.swing.*;

public class Clock extends JPanel implements ActionListener
{
    //创建时钟的外形
    protected static Ellipse2D face = new Ellipse2D.Float(3, 3, 94, 94);
    //创建时钟的标记
    protected static GeneralPath tick = new GeneralPath();
    static
    {
        tick.moveTo(100, 100);
        tick.moveTo(49, 0);
        tick.lineTo(51, 0);
        tick.lineTo(51, 6);
        tick.lineTo(49, 6);
        tick.lineTo(49, 0);
    }
    //创建时针
    protected static GeneralPath hourHand = new GeneralPath();
    static
    {
        hourHand.moveTo(50, 15);
        hourHand.lineTo(53, 50);
        hourHand.lineTo(50, 53);
        hourHand.lineTo(47, 50);
        hourHand.lineTo(50, 15);
    }
    //创建分针
    protected static GeneralPath minuteHand = new GeneralPath();
    static
    {
        minuteHand.moveTo(50, 2);
        minuteHand.lineTo(53, 50);
```

```
    minuteHand.lineTo(50, 58);
    minuteHand.lineTo(47, 50);
    minuteHand.lineTo(50, 2);
}
 //创建秒针
protected static GeneralPath secondHand = new GeneralPath();
static
{
    secondHand.moveTo(49, 5);
    secondHand.lineTo(51, 5);
    secondHand.lineTo(51, 62);
    secondHand.lineTo(49, 62);
    secondHand.lineTo(49, 5);
}
 //设置时钟的颜色
protected static Color faceColor = new Color(220, 220, 220);
protected static Color hourColor = Color.red.darker();
protected static Color minuteColor = Color.blue.darker();
protected static Color secondColor = new Color(180, 180, 0);
protected static Color pinColor = Color.gray.brighter();
//创建时钟的枢纽
protected Ellipse2D pivot = new Ellipse2D.Float(47, 47, 6, 6);
protected Ellipse2D centerPin = new Ellipse2D.Float(49, 49, 2, 2);

//创建绕时钟枢纽转的变换
protected AffineTransform hourTransform =
            AffineTransform.getRotateInstance(0, 50, 50);
protected AffineTransform minuteTransform =
            AffineTransform.getRotateInstance(0, 50, 50);
protected AffineTransform secondTransform =
            AffineTransform.getRotateInstance(0, 50, 50);
//创建每秒触发一次的 Timer
protected Timer timer = new Timer(1000, this);
protected Calendar calendar = Calendar.getInstance();

public Clock()
{
    setPreferredSize(new Dimension(100, 100));
}
 //当 plane 加入 container 中时发生
public void addNotify()
{
    super.addNotify();
    timer.start();
}
 //当 plane 从 container 中移除时发生
public void removeNotify()
{
    timer.stop();
    super.removeNotify();
}

public void actionPerformed(ActionEvent event)
{
    //更新 calender 的时间
    this.calendar.setTime(new java.util.Date());
    int hours = this.calendar.get(Calendar.HOUR);
    int minutes = this.calendar.get(Calendar.MINUTE);
    int seconds = this.calendar.get(Calendar.SECOND);

    //设置变换，使得时针、分针、秒针各自绕枢纽旋转一定的角度
    hourTransform.setToRotation((double) hours) *
                                (Math.PI / 6.0), 50, 50);
    minuteTransform.setToRotation((double) minutes) *
                                (Math.PI / 30.0), 50, 50);
    secondTransform.setToRotation((double) seconds) *
                                (Math.PI / 30.0), 50, 50);
    repaint();
```

```
    }

    public void paint(Graphics g)
    {
        super.paint(g);

        //得到图形上下文和抗锯齿处理
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setPaint(faceColor);
        g2.fill(face);
        g2.setPaint(Color.black);
        g2.draw(face);

        //产生钟面上12个滴答位置
        for (double p = 0.0; p < 12.0; p += 1.0)
        {
            //利用变换画出同心的滴答的标线
            g2.fill(tick.createTransformedShape(
                AffineTransform.getRotateInstance((Math.PI / 6.0) * p, 50, 50)));
        }
        g2.setPaint(hourColor);
        g2.fill(hourHand.createTransformedShape(hourTransform));
        g2.setPaint(minuteColor);
        g2.fill(minuteHand.createTransformedShape(minuteTransform));
        g2.setPaint(secondColor);
        g2.fill(secondHand.createTransformedShape(secondTransform));
        g2.fill(pivot);
        g2.setPaint(pinColor);
        g2.fill(centerPin);
    }

    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setLocation(700, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new Clock());
        frame.pack();
        frame.show();
    }
}
```

【运行结果】

运行结果如图 6-8 所示。



图 6-8

【分析与总结】

本程序主要的难点在于时钟的各个指针如何正确指示时间，这里用到了变换，用 Timer 每秒触发一次重画操作，从 Calendar 中得到当前的时间，再由 java.awt.geom 包中的 AffineTransform 类的 getRotateInstance(double theta, double x, double y)方法（x 和 y 表示旋转的中枢）确定旋转的起始位置，然后时针、分针和秒针分别按到达当前时间的目标位置应该旋转的角度进行旋转变换 setToRotation(double theta, double x, double y)，最后由 GeneralPath 的 createTransformedShape 方法返回一个 shape 供 Graphics2D 显示或填充显示。

实例72 椭圆形按钮

本实例演示如何用Java 2D创建更加美观的组件。

【编程要点】

需要改变JButton组件的外观，得继承一个BasicUI类，然后在子类中覆写paint(Graphics g, JComponent c)方法。

AbstractButton类中含有按钮和菜单项的一般方法，如getHeight()、getWidth()得到按钮的宽和高。

ButtonModel类中含有Check Box、Radio buttons或一般的buttons的状态，如isPressed()判断该组件是否被鼠标按下，isSelected()判断该组件是否被选中。用ButtonModel类可以设置在当按钮按下时显示不一样的外观。

【程序实现】

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.plaf.basic.*;
import javax.swing.plaf.*;

public class EllipseButton extends JButton
{
    public EllipseButton(String text)
    {
        super(text);
        //创建自己的UI
        this.setUI(EllipseButtonUI.createUI(this));
        //取消按钮的边框
        this.setBorder(null);
        //不用内容窗格的背景
        this.setContentAreaFilled(false);
        //设定放置椭圆的空间
        this.setMargin(new Insets(8, 14, 8, 14));
    }

    public static void main(String [] args)
    {
        //在UIManager中放置按钮字体
        UIManager.put("Button.font", new Font("Arial", 0, 20));
        JFrame frame = new JFrame();
        frame.setLocation(100, 100);
        frame.setSize(200, 80);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("Elliptical Button");
        frame.getContentPane().setLayout(new GridBagLayout());
        EllipseButton button = new EllipseButton("Elliptical");
        frame.getContentPane().add(button);
        frame.getContentPane().setBackground(new Color(230, 230, 150));
        frame.setVisible(true);
    }
}

//从BasicButtonUI中继承，该类适合所有的平台
class EllipseButtonUI extends BasicButtonUI
{
    protected static EllipseButtonUI singleton = new EllipseButtonUI();
    //创建Stroke用于画按钮
    protected static Stroke thickStroke = new BasicStroke(2.0f);

    public static ComponentUI createUI(JComponent c)
    {
        return singleton;
```

```
}

public void paint(Graphics g, JComponent c)
{
    //获得 Graphics2D 的对象，并开启抗锯齿处理
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
    //得到按钮的大小
    AbstractButton b = (AbstractButton) c;
    Rectangle viewRect = new Rectangle();
    viewRect.x = 0;
    viewRect.y = 0;
    viewRect.width = b.getWidth() - 1;
    viewRect.height = b.getHeight() - 1;
    //缩小矩形以显示抗锯齿效果
    viewRect.grow(-2, -2);
    //在按钮矩形内创建椭圆
    Ellipse2D ellipse = new Ellipse2D.Float();
    ellipse.setFrame(viewRect.getX(), viewRect.getY(),
            viewRect.getWidth(), viewRect.getHeight());
    //判断按钮有没有被按下
    ButtonModel model = b.getModel();
    boolean pressed = (model.isArmed() && model.isPressed()) ||
            model.isSelected();
    //根据按钮按下与否设置画笔颜色
    if (pressed)
    {
        Color background = UIManager.getColor("Button.select");
        g2.setPaint(background == null ? Color.gray : background);
    }
    else
        g2.setPaint(UIManager.getColor("control"));
    //填充椭圆按钮
    g2.fill(ellipse);
    //根据椭圆按钮的大小来设定边框大小.
    Arc2D arc = new Arc2D.Float();
    arc.setFrame(viewRect.getX(), viewRect.getY(),
            viewRect.getWidth(), viewRect.getHeight());
    arc.setArcType(Arc2D.OPEN);
   //设定边框指定弧度的区域
    arc.setAngles(viewRect.getWidth(), 0, 0, viewRect.getHeight());
    g2.setStroke(thickStroke);
    //根据按钮按下与否设定画笔的颜色
    g2.setPaint(pressed ?
           UIManager.getColor("controlDkShadow") :
           UIManager.getColor("controlHighlight"));
    g2.draw(arc);

    arc.setAngles(0, viewRect.getHeight(), viewRect.getWidth(), 0);
    g2.setPaint(pressed ?
           UIManager.getColor("controlHighlight") :
           UIManager.getColor("controlShadow"));
    g2.draw(arc);
    super.paint(g, c);
    //将图形上下文恢复原来的抗锯齿属性的设置
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_OFF);
}

public Dimension getPreferredSize(JComponent c)
{
    AbstractButton b = (AbstractButton) c;
    Dimension dim = super.getPreferredSize(c);
    //调整高度和宽度，为了更好的显示效果
    dim.height += (b.getMargin().top + b.getMargin().bottom);
    dim.width += (b.getMargin().left + b.getMargin().right);
```

```
        return dim;
    }
}
```

【运行结果】

按钮按下时如图 6-9 所示。

按钮未按下时如图 6-10 所示。



图 6-9



图 6-10

【分析与总结】

从本例可以看到，可以用 Java 2D 为更多的组件打造更加美观的外表。

实例 73　裁剪效果

本实例演示如何实现裁剪效果。

【编程要点】

在 Applet 中获取一个图像文件，可以调用 Applet 的 getImage()方法。但是 getImage 方法会在调用后马上返回，如果此时马上使用 getImage 获取的 Image 对象，而这时 Image 对象并没有真正装载或者装载完成。所以，在使用图像文件时，使用 java.awt 包中的 MediaTracker 跟踪一个 Image 对象的装载，可以保证所有图片都加载完毕。使用 MediaTracker 需要如下三个步骤：

（1）实例化一个 MediaTracker，注意要将显示图片的 Component 对象作为参数传入。

```
MediaTracker tracker = new MediaTracker(this);
```

（2）将要装载的 Image 对象加入 MediaTracker。

```
pics[i] = getImage(getCodeBase(),new Integer(i).toString()+".jpg");
tracker.addImage(pics[i], 0);
```

（3）调用 MediaTracker 的 checkAll()方法，等待装载过程的结束。

“tracker.checkAll(true);”或者用 tracker.waitForID(0)等待标记为 0 的图片装载结束。

设置裁剪属性时，可调用 Graphics2D.setClip 并传递将用来作为 clipping path 的 Shape，还可以通过调用 clip 方法缩小 clipping path 并传递另外一个 Shape。裁剪区将是当前裁剪和特定 shape 的交集区。本实例裁剪一个 image，clipping path 是一个椭圆和一个随机大小的矩形的交集区。椭圆被传递给 setClip 方法，然后调用 clip 方法设置 clipping path 为椭圆和矩形的交集区。

【程序实现】

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.awt.event.WindowAdapter;
import java.awt.image.BufferedImage;
import java.net.URL;

public class ClipImage extends JApplet implements Runnable {
    private Image img;
    private final double OINC[] = { 5.0, 3.0 };
    private final double SINC[] = { 5.0, 5.0 };
    private double x, y;
    private double ix = OINC[0];
```

```
private double iy = OINC[1];
private double iw = SINC[0];
private double ih = SINC[1];
private double ew, eh; //椭圆的宽和高
private GeneralPath p = new GeneralPath();
private AffineTransform at = new AffineTransform();
private BasicStroke bs = new BasicStroke(20.0f);
private Arc2D arc = new Arc2D.Float();
private Ellipse2D ellipse = new Ellipse2D.Float();
private RoundRectangle2D roundRect = new RoundRectangle2D.Float();
private Rectangle2D rect = new Rectangle2D.Float();
private Color redBlend = new Color(255, 0, 0, 120);
private Color greenBlend = new Color(0, 255, 0, 120);
private Thread thread;
private BufferedImage offImg;
private int w, h;
private boolean newBufferedImage;

public void init() {
    setBackground(Color.white);
    img = getImage(getURL("images/Hill.jpg"));
    try {
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(img, 0);
        tracker.waitForID(0);
    } catch (Exception e) {
    }
}

public void drawDemo(Graphics2D g2) {
    if (newBufferedImage) {
        x = Math.random() * w;
        y = Math.random() * h;
        ew = (Math.random() * w) / 2;
        eh = (Math.random() * h) / 2;
    }
    x += ix;
    y += iy;
    ew += iw;
    eh += ih;
    if (ew > w / 2) {
        ew = w / 2;
        iw = Math.random() * -w / 16 - 1;
    }
    if (ew < w / 8) {
        ew = w / 8;
        iw = Math.random() * w / 16 + 1;
    }
    if (eh > h / 2) {
        eh = h / 2;
        ih = Math.random() * -h / 16 - 1;
    }
    if (eh < h / 8) {
        eh = h / 8;
        ih = Math.random() * h / 16 + 1;
    }
    if ((x + ew) > w) {
        x = (w - ew) - 1;
        ix = Math.random() * -w / 32 - 1;
    }
    if (x < 0) {
        x = 2;
        ix = Math.random() * w / 32 + 1;
    }
    if ((y + eh) > h) {
        y = (h - eh) - 2;
        iy = Math.random() * -h / 32 - 1;
    }
    if (y < 0) {
        y = 2;
        iy = Math.random() * h / 32 + 1;
```

```
        }

        ellipse.setFrame(x, y, ew, eh);
        g2.setClip(ellipse);

        rect.setRect(x + 5, y + 5, ew - 10, eh - 10);
        g2.clip(rect);

        g2.drawImage(img, 0, 0, w, h, this);

        p.reset();
        p.moveTo(-w / 2.0f, -h / 8.0f);
        p.lineTo(+w / 2.0f, -h / 8.0f);
        p.lineTo(-w / 4.0f, +h / 2.0f);
        p.lineTo(+0.0f, -h / 2.0f);
        p.lineTo(+w / 4.0f, +h / 2.0f);
        p.closePath();

        at.setToIdentity();
        at.translate(w * .5f, h * .5f);
        g2.transform(at);
        g2.setStroke(bs);
        g2.setPaint(redBlend);
        g2.draw(p);

        at.setToIdentity();
        g2.setTransform(at);

        g2.setPaint(greenBlend);

        for (int yy = 0; yy < h; yy += 50) {
            for (int xx = 0, i = 0; xx < w; i++, xx += 50) {
                switch (i) {
                case 0:
                    arc.setArc(xx, yy, 25, 25, 45, 270, Arc2D.PIE);
                    g2.fill(arc);
                    break;
                case 1:
                    ellipse.setFrame(xx, yy, 25, 25);
                    g2.fill(ellipse);
                    break;
                case 2:
                    roundRect.setRoundRect(xx, yy, 25, 25, 4, 4);
                    g2.fill(roundRect);
                    break;
                case 3:
                    rect.setRect(xx, yy, 25, 25);
                    g2.fill(rect);
                    i = -1;
                }

            }
        }
    }

    public Graphics2D createDemoGraphics2D(Graphics g) {
        Graphics2D g2 = null;

        if (offImg == null || offImg.getWidth() != w || offImg.getHeight() != h) {
            offImg = (BufferedImage) createImage(w, h);
            newBufferedImage = true;
        }

        if (offImg != null) {
            g2 = offImg.createGraphics();
            g2.setBackground(getBackground());
        }

        //设置属性
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
```

```
        g2.setRenderingHint(RenderingHints.KEY_RENDERING,
                RenderingHints.VALUE_RENDER_QUALITY);

        //清除画布
        g2.clearRect(0, 0, w, h);

        return g2;
    }

    public void paint(Graphics g) {

        w = getWidth();
        h = getHeight();

        if (w <= 0 || h <= 0)
            return;
      //创建整个图片显示
        Graphics2D g2 = createDemoGraphics2D(g);
        //在上面图片的基础上裁剪
        drawDemo(g2);
        g2.dispose();

        if (offImg != null && isShowing()) {
            g.drawImage(offImg, 0, 0, this);
        }

        newBufferedImage = false;
    }

    public void start() {
        thread = new Thread(this);
        thread.start();
    }

    public synchronized void stop() {
        thread = null;
    }

    public void run() {

        Thread me = Thread.currentThread();

        while (thread == me && isShowing()) {
            Graphics g = getGraphics();
            paint(g);
            g.dispose();
            thread.yield();
        }
        thread = null;
    }

    protected URL getURL(String filename) {
        URL codeBase = this.getCodeBase();
        URL url = null;

        try {
            url = new URL(codeBase, filename);
        } catch (java.net.MalformedURLException e) {
            System.out.println("Couldn't create image: " + "badly specified URL");
            return null;
        }

        return url;
    }

    public static void main(String s[]) {
        final ClipImage demo = new ClipImage();
        WindowListener l = new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
```

```
            }

            public void windowDeiconified(WindowEvent e) {
                demo.start();
            }
            public void windowIconified(WindowEvent e) {
                demo.stop();
            }
        };
        JFrame f = new JFrame("Java 2D Demo - ClipImage");
        f.addWindowListener(l);
        f.getContentPane().add("Center", demo);
        f.setSize(new Dimension(400, 300));
        f.show();
        demo.start();
    }
}
```

【运行结果】

运行结果如图 6-11 所示。



图 6-11

【分析与总结】

由程序可以看出，要定义剪切路径：先创建一个 Shape，代表想要绘制的区域。再调用 Graphics2D.setClip，将该形状用作 Graphics2D 上下文的剪切路径。要缩小剪切路径：创建一个与当前剪切路径相交的 Shape。调用 clip，将剪切路径更改为当前剪切路径与新 Shape 相交的部分。另外裁剪区域还可以是文本，从而可以实现艺术字的效果。

实例 74　Applet 音频播放器

在 Java 2 平台出现之前，Java 语言只能处理电话音质的声音，以单声道 8KHz 的采样频率存储为 μ-law AU 文件。Java 2 平台增加了对 AIFF、WAV 以及三种 MIDI 文件类型的支持。所支持的三种 MIDI 文件格式为 MIDI 文件类型 0、MIDI 文件类型 1 以及 RMF。

主要的声音文件类型如下：

AU（扩展名为 AU 或 SND）：适用于短的声音文件，为 Solaris 和下一代机器的通用文件格式，也是 Java 平台的标准的音频格式。AU 类型文件使用的三种典型音频格式为：8 位 μ-law 类型（通常采样频率为 8KHz），8 位线性类型以及 16 位线性类型。

WAV（扩展名为 WAV）：由 Microsoft 和 IBM 共同开发，对 WAV 的支持已经被加进 Windows 95 并且被延伸到 Windows 98。WAV 文件能存储各种格式包括 μ-law、a-law 和 PCM（线性）数据。它们几乎能被所有支持声音的 Windows 应用程序播放。

AIFF（扩展名为 AIF 或 IEF）音频互换文件格式是为 Macintosh 计算机和 Silicon Graphics（SGI）计算机所共用的标准音频文件格式。AIFF 和 AIFF-C 几乎是相同的，除了后者支

持例如 μ-law 和 IMA ADPCM 类型的压缩。

MIDI（扩展名为 MID）乐器数字接口 MIDI 是为音乐制造业所认可的标准，主要用于控制诸如合成器和声卡之类的设备。

Java 声音引擎支持两种 MIDI 文件类型：

MIDI 类型 0 文件 - 包含仅仅一个序列，所有相关的乐器部分被包含在同一个逻辑“磁道”上。

MIDI 类型 1 文件 - 包含多重的“磁道”使得不同的乐器被逻辑地分开，从而使对声音的操作和重组更加容易。

RMF（扩展名为 RMF）混合音乐格式是由 Beatnik 设计出来的混合文件类型，通过交互式设定将 MIDI 和音频采样封装在一起。RMF 好比是一个所有音乐相关文件的容器。RMF 也包含对有关版权的详细文件说明的支持。RMF 文件可以包含多个由不同艺术家创作的存储为 MIDI 类型或音频采样类型的作品，每个都关联着相关的版权信息。

Java 中用于播放声音的 AudioClip 接口提供下列机能：播放、循环、停止。恢复和播放声音最简单的方法是通过 Applet 类的 play()方法。调用 play()方法有如下两种方式：

play()带有一个参数，一个 URL 对象，装载并演奏储存在 URL 处的声音片断。play()带有两个参数，基本 URL 和文件夹路径名，装载并演奏声音文件。第一参数经常是对 getCodeBase()或 getDocumentBase()的调用。

play()一旦被调用立刻开始恢复和播放声音。如果声音文件不能被查找，将不会有出错信息，仅仅是沉默。启动和停止声音文件，或循环播放，必须用 applet 的 getAudioClip 方法把它装载进入 AudioClip 对象，类似上面 play()方法的参数，getAudioClip 方法要用一个或两个参数，当作播放的指示。第一个或惟一的一个参数是 URL 参数，用来指示声音文件的位置，第二参数是文件夹路径指针。

【程序实现】

```
//SoundApplet.java
import javax.swing.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class SoundApplet extends JApplet
                    implements ActionListener,
                               ItemListener {
    AppletSoundList soundList;
    String auFile = "sound\\spacemusic.au";
    String aifffile = "sound\\flute+hrn+mrmba.aif";
    String midiFile = "sound\\trippygaia1.mid";
    String rmfFile = "sound\\jungle.rmf";
    String wavFile = "sound\\bottle-open.wav";
    String chosenFile;
    AudioClip onceClip, loopClip;
    JComboBox formats;
    JButton playButton, loopButton, stopButton;
    boolean looping = false;
    public void init() {
        String [] fileTypes = {auFile,
                               aiffFile,
                               midiFile,
                               rmfFile,
                               wavFile};
        formats = new JComboBox(fileTypes);
        formats.setSelectedIndex(0);
```

```
        //获得选中的声音文件名
        chosenFile = (String)formats.getSelectedItem();
        formats.addItemListener(this);
        playButton = new JButton("Play");
        playButton.addActionListener(this);
        loopButton = new JButton("Loop");
        loopButton.addActionListener(this);
        stopButton = new JButton("Stop");
        stopButton.addActionListener(this);
        stopButton.setEnabled(false);

        JPanel controlPanel = new JPanel();
        controlPanel.add(formats);
        controlPanel.add(playButton);
        controlPanel.add(loopButton);
        controlPanel.add(stopButton);
        getContentPane().add(controlPanel);
        startLoadingSounds();
    }

    public void itemStateChanged(ItemEvent e) {
        chosenFile = (String)formats.getSelectedItem();
        soundList.startLoading(chosenFile);
    }
    void startLoadingSounds() {
        //开始装载声音文件
        soundList = new AppletSoundList(this, getCodeBase());
        soundList.startLoading(auFile);
        soundList.startLoading(aiffFile);
        soundList.startLoading(midiFile);
        soundList.startLoading(rmfFile);
        soundList.startLoading(wavFile);
    }
    public void stop() {
        onceClip.stop();          //停止一次播放
        if (looping) {
            loopClip.stop();      //停止循环播放
        }
    }
    public void start() {
        if (looping) {
            loopClip.loop();                    //重新启动循环播放
        }
    }
    public void actionPerformed(ActionEvent event) {
        Object source = event.getSource();
        if (source == playButton) {
            //Try to get the AudioClip.
            onceClip = soundList.getClip(chosenFile);
            onceClip.play();    //Play it once.
            stopButton.setEnabled(true);
            showStatus("Playing sound " + chosenFile + ".");
            if (onceClip == null) {
                showStatus("Sound " + chosenFile + " not loaded yet.");
            }
            return;
        }
        //如果按下循环按钮
        if (source == loopButton) {
            loopClip = soundList.getClip(chosenFile);
            looping = true;
            loopClip.loop();                    //开始循环播放
            loopButton.setEnabled(false);
            stopButton.setEnabled(true);
            showStatus("Playing sound " + chosenFile + " continuously.");
            if (loopClip == null) {
                showStatus("Sound " + chosenFile + " not loaded yet.");
            }
            return;
```

```
        }
        //如果按下停止按钮
        if (source == stopButton) {
            if (looping) {
                looping = false;
                loopClip.stop();                    //停止循环播放
                loopButton.setEnabled(true);        //使开始按钮可用
            }
            else if (onceClip != null) {
                onceClip.stop();
            }
            stopButton.setEnabled(false);
            showStatus("Stopped playing " + chosenFile + ".");
            return;
        }
    }
}

//AppletSoundList.java
import javax.swing.*;
import java.applet.*;
import java.net.URL;

//Loads and holds a bunch of audio files whose locations are specified
//relative to a fixed base URL.
class AppletSoundList extends java.util.Hashtable {
    JApplet applet;
    URL baseURL;
    public AppletSoundList(JApplet applet, URL baseURL) {
        super(5); //初始化哈希表的容量为 5
        this.applet = applet;
        this.baseURL = baseURL;
    }
    public void startLoading(String relativeURL) {
        new AppletSoundLoader(applet, this,
                              baseURL, relativeURL);
    }
    //通过相对 URL 从哈希表中获得 AudioClip
    public AudioClip getClip(String relativeURL) {
        return (AudioClip)get(relativeURL);
    }
    //将 AudioClip 对象放进哈希表，键值为 relativeURL
    public void putClip(AudioClip clip, String relativeURL) {
        put(relativeURL, clip);
    }
}

//AppletSoundLoader.java
import javax.swing.*;
import java.applet.*;
import java.net.URL;
class AppletSoundLoader extends Thread {
    JApplet applet;
    AppletSoundList soundList;
    URL baseURL;
    String relativeURL;
    public AppletSoundLoader(JApplet applet,
                             AppletSoundList soundList,
                             URL baseURL,
                             String relativeURL) {
        this.applet = applet;
        this.soundList = soundList;
        this.baseURL = baseURL;
        this.relativeURL = relativeURL;
        setPriority(MIN_PRIORITY);
        start();
    }
    public void run() {
        AudioClip audioClip = applet.getAudioClip(baseURL, relativeURL);
```

```
            soundList.putClip(audioClip, relativeURL);
        }
    }
```

【运行结果】

运行结果如图 6-12 所示。



图 6-12

【分析与总结】

本程序有三个文件：SoundApplet.java 是主文件，AppletSoundLoader.java 中的 AppletSoundLoader 类继承 Thread 负责为声音文件获得 AudioClip 对象并把它加到 AppletSoundList 中去；AppletSoundList.java 中定义的 AppletSoundList 继承自 Hashtable，提供 getClip(String)和 putClip(AudioClip,String)方法，分别从哈希表中取得 AudioClip 对象和放置 AudioClip 对象。其结构是 AppletSoundList 中用 AppletSoundLoader 来装载指定 URL 的声音文件，AppletSoundLoader 负责为声音文件产生 AudioClip 对象并添加到 AppletSoundList 中。在 SoundApplet 中则只是利用 AppletSoundList 类。

实例 75　音频播放

本实例演示如何用 Sound API 进行音频播放操作。

【编程要点】

JavaSound 是一个小巧的低层 API，支持数字音频和 MIDI 数据的记录/回放。在 JDK 1.3.0 之前，JavaSound 是一个标准的 Java 扩展 API，但从 Java 2 的 1.3.0 版开始，JavaSound 就被包含到 JDK 之中。由于 Java 有着跨平台（操作系统、硬件平台）的特点，基于 JavaSound 的音频处理程序（包括本文的程序）能够在任何实现了 Java 1.3+的系统上运行，无需加装任何支持软件。

javax.media.sound.midi：该程序包提供 MIDI 合成、序列化以及事件传送的界面。MidiSystem 类充当 MIDI 音乐系统入口的角色，它提供相关设备的信息并对之进行访问，这些设备包括传送器、接收器、合成器以及音序器。

基本的 MIDI 传送界面是 MidiDevice。所有设备为列出其所支持的模式和查询当前模式提供了方法。设备支持监听器，可以监听各种事件（如打开和关闭事件），并且有一个信息对象来描述设备状态。通常，设备是 MIDI 事件的传送器或接收器之一。传送器界面提供了一些方法来设置和查询接收器，该接收器接收由该传送器发送出的 MIDI 事件。接收器为接收 MIDI 事件提供方法。基本的 MidiEvent 对象是通过一个消息说明事件类型、数据长度以及状态。它也为涉及 MIDI 计时的设备（例如音序器）提供准确的计时信号。

合成器界面是一种生成声音的特殊类型的接收器。它也提供一些方法来管理声音库和

乐器。另外，合成器可以支持一组全局的非 MIDI 控件（例如增益和定位）。它也提供对一组 MIDI 通道的访问，实际上，声音就是通过这些通道产生的。

MidiChannel 界面提供一组方法来表达公用的 MIDI 声音消息（例如 noteON、noteOff 以及 controlChange）。还支持对当前通道状态的查询。

序列化的 Sequencer 对象继承自 MidiDevice，它提供了完成基本 MIDI 序列化操作的方法。音序器可以装载和播放一个序列，查询和设置节奏，并且控制主从同步方式。一个应用程序还可以通过注册使得音序器在处理设备单元和控制器事件时通知自己。序列化对象代表一个来自一个或多个磁道的 MIDI 序列以及相关的计时信息。一个磁道对象包含一系列加上时间标记的 MIDI 事件。

Java 声音 API 在 MidiSystem 类中为在 MIDI 文件和序列化对象之间转换提供了高级的接口。

【程序实现】

```
import javax.sound.midi.*;
import java.io.*;
import java.net.*;

public class MidiDemo {
    private static String midiFile = "sound\\trippygaia1.mid";
    //private static String midiURI = "http://hostname/midifile";
    private Sequence sequence = null;
    public MidiDemo() {
        this.loadAndPlay();
    }
    public void loadAndPlay() {
        try {
            //从文件中读取
            sequence = MidiSystem.getSequence(new File(midiFile));
            //从 URL 中读取
            //sequence = MidiSystem.getSequence
            //            (new URL("http://hostname/midifile"));
            //为 midi 序列创建一个 sequencer 对象
            Sequencer sequencer = MidiSystem.getSequencer();
            sequencer.open();
            sequencer.setSequence(sequence);
            //计算 midi 播放持续的时间
            double durationInSecs = sequencer.getMicrosecondLength() / 1000000.0;
            System.out.println("the duration of this audio is "
                    + durationInSecs + "secs.");
            //计算 Midi Sequencer 中播放的当前位置
            double seconds = sequencer.getMicrosecondPosition() / 1000000.0;
            System.out.println("the Position of this audio is " + seconds + "secs.");
            //设置 Midi 声音播放的音量
            if (sequencer instanceof Synthesizer) {
                Synthesizer synthesizer = (Synthesizer) sequencer;
                MidiChannel[] channels = synthesizer.getChannels();
                //gain 的值在 0 和 1 之间(loudest)
                double gain = 0.9D;
                for (int i = 0; i < channels.length; i++) {
                    channels[i].controlChange(7, (int) (gain * 127.0));
                }
            }
            //开始播放
            sequencer.start();
            //计算当前的 Midi Sequencer 播放位置
            Thread.currentThread().sleep(5000);
            seconds = sequencer.getMicrosecondPosition() / 1000000.0;
            System.out.println("the Position of this audio is " + seconds
                    + "secs.");
```

```
                //添加媒体信息的监听器
                sequencer.addMetaEventListener(new MetaEventListener() {
                    public void meta(MetaMessage event) {
                        if (event.getType() == 47) {
                            System.out.println("Sequencer is done playing.");
                        }
                    }
                });
            } catch (MalformedURLException e) {
            } catch (IOException e) {
            } catch (MidiUnavailableException e) {
            } catch (InvalidMidiDataException e) {
            } catch (InterruptedException e) {
            }
        }
        public static void main(String[] args) {
            MidiDemo midi = new MidiDemo();
        }
    }
```

【运行结果】

可以听到声音播放，并且在控制台输出：

```
the duration of this audio is 66.229424secs.
the Position of this audio is 0.0secs.
the Position of this audio is 5.01552secs.
Sequencer is done playing.
```

【分析与总结】

从上面的程序可以看出，播放 MIDI 声音文件时，先从文件中得到 sequence，生成音序器（sequencer），通过 open()打开设备，把 squence 通过 setSequence()装载到 sequencer。将 sequence 转化为 synthesizer，然后在 synthesizer 中用 getChannels 方法得到各个通道，在 Channel 类的方法 controlChange 中可以调节音量。

在 sequencer 上添加媒体事件监听器，可以在声音文件结束后在控制台输出结束的信息：Sequencer is done playing。

小结

本章介绍了 Java 2D API 的几个方法，Java 2D API 为 Java 开发人员创建复杂的 2D 图形、文字及图像提供了方法，支持复杂、灵活且不依赖于设备和显示分辨率的图形框架结构。

另外，本章还介绍了 AudioClip 类和 javax.media.midi 包 MidiSystem 类实现的音频播放。如果需要更多更全面的多媒体编程，那么需要到 Sun 主页下载 JMF API，JMF 为媒体播放器、媒体的获取以及媒体会议定义了统一的体系结构、消息处理协议和程序设计接口。

第 7 章　Java 网络编程

本章通过 11 个实例介绍了 Java 的网络编程，其中包括访问并获得 URL、UDP 传送、代理服务器、网页浏览器、Telnet、扫描应用端口、聊天工具、上传文件、IP 多组传播、邮件发送、FTP 浏览器等，网络编程是 Java 编程的重头戏，读者应努力掌握它。

实例 76　访问并获得 URL 资源

本实例通过 URL 类和它的方法来下载 URL 的内容，并把它放入文件中。

【编程要点】

URL 类表示一个标准的资源定位器，一个指向 WWW“资源”的指针。一个资源有时可能像一个文件或目录一样简单，也可能是一个更为复杂的对象的引用，例如一个数据库的查询或一个搜索引擎。

构造函数有：

URL(String)：从 String 生成一个 URL 对象。

URL(String, String, int, String)：从指定的 protocol、host、port 号码和 file 生成一个 URL 对象。

URL(String, String, String)：从指定 protocol 名、host 名和 file 名生成一个绝对的 URL。

URL(URL, String)：并在指定的上下文里分析规范 spec 而生成一个 URL。

含有的方法有：

openStream()：打开到 URL 的连接，并返回用于从该连接读取数据的 InputStream。

getHost()：如果适用，返回 URL 的主机名。

getPort()：返回 URL 的端口号。

【程序实现】

```
import java.io.*;
import java.net.*;

public class GetURL {
    public static void main(String[] args) {
        InputStream in = null;
        OutputStream out = null;
        try {
            //检查命令行参数
            if ((args.length != 1)&& (args.length != 2))
                throw new IllegalArgumentException("Wrong number of args");

            //建立输入/输出流
            URL url = new URL(args[0]);
            in = url.openStream();
            if (args.length == 2)
                out = new FileOutputStream(args[1]);
            else out = System.out;
            //从 URL 拷贝到输出流
            byte[] buffer = new byte[4096];
            int bytes_read;
            while((bytes_read = in.read(buffer)) != -1)
                out.write(buffer, 0, bytes_read);
    }
```

```
            //捕获异常、输出报错信息
            catch (Exception e) {
                System.err.println(e);
                System.err.println("Usage: java GetURL <URL> [<filename>]");
            }
            finally {  //保证关闭输入流和输出流
                try { in.close();  out.close(); } catch (Exception e) {}
            }
        }
    }
```

【运行结果】

在控制台输入 java GetURL http://www.eclipse.org D:\test.txt，那么该网页上的 HTML 文件的内容会下载到 D:\test.txt 文件中。

实例 77 UDP 传送

本实例利用 UDP（用户数据报协议）传送文件或字符串到指定主机和端口。

【编程要点】

数据报的客户端和服务器端的类在表面上是一样的，UDP 协议是无连接的。

InetAddress 类表示一个 Internet Protocol(IP)地址。应用程序必须使用方法 getLocalHost、getByName 或 getAllByName 来创建新的 InetAddress 实例：

getByName(String host)，给定的主机名，确定该主机的所有 IP 地址。主机名可以是机器名，如“java.sun.com”，也可以是主机 IP 地址的字符串表示，如“206.26.48.100”。host：指定的主机，或本地主机（null）。返回给定主机名的 IP 地址，如果找不到该 host 的 IP 地址抛出：UnknownHostException。

getLocalHost()，返回本地主机 IP 地址，如果找不到该 host 的 IP 地址则抛出 UnknownHostException。

getAllByName(String host)，给定主机名，确定该主机的所有 IP 地址。主机名可以是机器名，如“java.sun.com”，也可以是主机 IP 地址的字符串表示，如“206.26.48.100”。参数 host 表示主机名，返回给定主机名的 IP 地址数组。如果找不到该 host 的 IP 地址抛出 UnknownHostException。

DatagramPacket 此类表示一个数据报文包。该数据报文包用来实现一个无连接包的传送服务。每条消息从一台机器发送到另一台机器的路径仅依赖于这个包所包含的信息。多个包从一台机器发送到另一台机器可能经过不同的路径，并且以任意顺序到达。构造函数有：

DatagramPacket(byte[], int)：构造 DatagramPacket 用来接收长度为 ilength 的包。

DatagramPacket(byte ibuf[], int ilength, InetAddress iaddr, int iport)：构造数据报文包，用它把长度为 ilength 的包传送到指定宿主的指定的端口号。length 自变量必须小于等于 ibuf.length。ibuf：报文数据；ilength：报文长度。laddr：目的地址。iport：目的地址的端口号。

其他方法：

public synchronized InetAddress getAddress()：返回接收或发送该数据报文机器的 IP 地址。

public synchronized int getPort()：返回接收或发送该数据报文的远程主机端口号。

public synchronized byte[] getData()：返回接收到的或发送出的数据。

public synchronized int getLength()：返回发送出的或接收到的数据的长度。

DatagramSocket 类表示一个发送和接收数据报文包的 socket。一个数据报文 socket 是无连接包传递服务的发送或接收点。每一个在数据报文 socket 发送或接收的包都独立地进行寻址和路由。多个包从一台机器到另一台机器可能经过不同的路径，并且可以以任意顺序到达。

DatagramSocket()：构造一个数据报文 socket 并且把它捆绑到本地主机任意可用的端口。

DatagramSocket(int port)：构造一个数据报文 socket 并且把它捆绑到本地主机指定的端口。

参数：port 指使用的端口。如果该 socket 无法打开，或该 socket 无法绑定到指定的本机端口抛出 SocketException。

DatagramSocket(int port, InetAddress laddr)生成一个数据报文 socket，并捆绑到指定的本地地址。本机端口必须为 0 和 65535 之间的一个。参数：port 指使用的本机端口，laddr 指捆绑的本地地址。

其他方法：

void send(DatagramPacket p)从此 socket 发送数据报文包。DatagramPacket 包含发送的数据的有关信息，包括它的长度，远程主机的 IP 地址和远程主机的端口号。

参数：p 为发送的 DatagramPacket。如果发生一个 I/O 错误抛出：IOException。

public synchronized void receive(DatagramPacket p)：从此 socket 接收数据报文包。当该方法返回时，接收的数据填充到 DatagramPacket 的缓冲区。该数据报文包还包含发送方的 IP 地址和发送方机器的端口号。该方法按块记录，直到接收到一个数据报文包。数据报文包对象的 length 字段包含接收的报文的长度。如果报文的长度大于缓冲区长度，报文将被截断。

参数：p 为输入的数据放入的 DatagramPacket。如果发生一个 I/O 错误则抛出：IOException。

public InetAddress getLocalAddress()：获取该 socket 绑定的本地地址。

public int getLocalPort()：返回该 socket 绑定的本地主机的端口号码。

public void close()：关闭该数据报文 socket。

public synchronized void setSoTimeout(int timeout)：以毫秒为单位，设置或取消 SO_TIMEOUT 指定的超时。使用设置成非零的超时选择项对该 DatagramSocket 调用 receive()方法，该方法在这段设定的时间内阻塞。

如果超时得以验证，java.io.InterruptedIOException 将被启动，即使 ServerSocket 仍有效。选择项 must 被设置成可以加入封锁操作。超时必须是> 0。取值为零的超时被解释为无穷大的超时。

public synchronized int getSoTimeout()：提取 SO_TIMEOUT 的设置。返回值为 0 表示该选择项是屏蔽的（如：超时为无穷大）。

【程序实现】

```
import java.io.*;
```

```
import java.net.*;

public class UDPSend {
    public static final String usage =
     "Usage: java UDPSend <hostname> <port> <msg>...\n" +
     "   or: java UDPSend <hostname> <port> -f <file>";

    public static void main(String args[]) {
        try {
            //检查参数的个数
            if (args.length < 3)
                throw new IllegalArgumentException("Wrong number of args");

            //解析参数
            String host = args[0];
            int port = Integer.parseInt(args[1]);
            byte[] message;                          //要传送的数据
            if (args[2].equals("-f")) {              //传送文件
                File f = new File(args[3]);
                int len = (int)f.length();           //文件长度
                message = new byte[len];             //建立缓冲区
                FileInputStream in = new FileInputStream(f);
                int bytes_read = 0, n;
                do {                                 //从文件中读取
                    n = in.read(message, bytes_read, len-bytes_read);
                    bytes_read += n;
                } while((bytes_read < len)&& (n != -1));
            }
            else {                                   //发送字符串
                String msg = args[2];
                for (int i = 3; i < args.length; i++) msg += " " + args[i];
                message = msg.getBytes();
            }

            //根据主机名称得到IP地址
            InetAddress address = InetAddress.getByName(host);
            //用数据和地址创建数据报文包
            DatagramPacket packet = new DatagramPacket(message, message.length,
                                    address, port);

            //创建数据报文套接字并通过它传送
            DatagramSocket dsocket = new DatagramSocket();
            dsocket.send(packet);
            dsocket.close();
        }
        catch (Exception e) {
            System.err.println(e);
         System.err.println(usage);
        }
    }
}
```

【运行结果】

在控制台输入 java UDPSend 主机名、端口号、要发送的信息，或者 java UDPSend 主机名、端口号、-f 文件名，就能将信息或文件发送到存在的主机的对应端口。

【分析与总结】

UDP 数据传送是不可靠的传送，所以无法知道传送是否被对方接收。一个数据报服务器的基本步骤是：

（1）在一个指定的端口上建立一个数据报 socket。

（2）用 receive 方法等待进来的包。

（3）用特定的协议来回应收到的包。

（4）回到第（2）步或继续第（2）步。

（5）关闭数据报 socket。

实例 78 代理服务器

本实例演示代理服务器的工作。

【编程要点】

代理服务器打开一个端口接收浏览器发来的访问某个站点的请求，从请求的字符串中解析出用户想访问哪个网页，然后通过 URL 对象建立输入流读取相应的网页内容，最后按照 Web 服务器的工作方式将网页内容发送给用户浏览器。

ServerSocket 类实现服务器 socket，一个服务器 socket 等待从网络上来的请求。它执行某些基于请求的操作，然后可能返回一个结果给请求者。服务器 socket 的实际工作是由一个 SocketImpl 类的实例执行的。

一个应用可以改变生成 socket 的 socket 工厂，用以配置自己以生成适合本地防火墙的 socket。

其中抽象类 SocketImpl 是所有实际实现 socket 的类的公共超类。它被用来生成客户机和服务器 socket。

【程序实现】

```
import java.net.*;
import java.io.*;

public class MyProxyServer {
    public static void main(String args[]) {
        try {
            ServerSocket ss = new ServerSocket(8080);
            System.out.println("proxy server OK");
            while (true) {
                Socket s = ss.accept();
                Process p = new Process(s);
                Thread t = new Thread(p);
                t.start();
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

class Process implements Runnable {
    Socket s;

    public Process(Socket s1) {
        s = s1;
    }

    public void run() {
        String content = " ";
        try {
            //从 Socket 中获得输入
            PrintStream out = new PrintStream(s.getOutputStream());
            BufferedReader in = new BufferedReader(new InputStreamReader(s
                    .getInputStream()));
            String info = in.readLine();
            //分析请求字符串，得到 URL
            System.out.println("now got " + info);
            int sp1 = info.indexOf(' ');
            int sp2 = info.indexOf(' ', sp1 + 1);
            String gotourl = info.substring(sp1, sp2);
```

```
                System.out.println("now connecting " + gotourl);
                URL con = new URL(gotourl);
                InputStream gotoin = con.openStream();
                int n = gotoin.available();
                byte buf[] = new byte[1024];
                out.println("HTTP/1.0 200 OK");
                out.println("MIME_version:1.0");
                out.println("Content_Type:text/html");
                out.println("Content_Length:" + n);
                out.println(" ");
                while ((n = gotoin.read(buf)) >= 0) {
                    out.write(buf, 0, n);
                }
                out.close();
                s.close();
            } catch (IOException e) {
                System.out.println("Exception:" + e);
            }
        }
    }
```

【运行结果】

控制台输出：proxy server OK。然后便开始实时监控浏览器发来的请求，分析出URL地址 now got（请求信息）now connecting（网址）最后是网页的内容。

【分析与总结】

在Process的run中从打开一个Socket端口中接收浏览器发来的访问某个站点的请求，从请求的字符串中解析出用户想访问哪个网页，让后通过URL对象建立输入流读取相应的网页内容并输出至控制台。

实例79 网页浏览器

本实例实现一个简单的网页浏览器。

【编程要点】

JEditorPane的setPage(URL url)方法来显示页面，只需从TextField中获URL地址，再在TextField添加事件监听器，使得在TextField中按下回车键时显示页面。

【程序实现】

```
import java.io.IOException;
import java.net.URL;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

public class WebBrowser extends JFrame {
    JPanel contentPane;                                    //包含整个框架的容器
    BorderLayout borderLayoutAll = new BorderLayout();
    JLabel jLabelPrompt = new JLabel();                    //状态提示框
    JPanel jPanelMain = new JPanel();
    BorderLayout borderLayoutMain = new BorderLayout();
    JTextField textFieldURL = new JTextField(); //URL输入框
    JEditorPane jEditorPane = new JEditorPane();//显示网页内容的容器

    public WebBrowser() {                                  //定义构造方法
        try {
            Init();                                        //初始化并显示界面
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    private void Init() throws Exception {        //界面初始化
        contentPane = (JPanel) getContentPane();
        contentPane.setLayout(borderLayoutAll);
        jPanelMain.setLayout(borderLayoutMain);
        jLabelPrompt.setText("请输入 URL");
        textFieldURL.setText("");                 //清空文本框
        textFieldURL.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                textFieldURL_actionPerformed(e);
            }
        });
        jEditorPane.setEditable(false);                    //设置不可编辑
        jEditorPane
                .addHyperlinkListener(new javax.swing.event.HyperlinkListener() {
                    public void hyperlinkUpdate(HyperlinkEvent e) {
                        jEditorPane_hyperlinkUpdate(e);
                    }
                });
        JScrollPane scrollPane = new JScrollPane();
        scrollPane.getViewport().add(jEditorPane);
        jPanelMain.add(textFieldURL, "North");
        jPanelMain.add(scrollPane, "Center");
        contentPane.add(jLabelPrompt, "North");
        contentPane.add(jPanelMain, "Center");
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        this.setSize(new Dimension(600, 500));
        this.setTitle("迷你 IE ");
        this.setVisible(true);
    }

    void textFieldURL_actionPerformed(ActionEvent e) {    //输入地址后响应回车
        try {
            jEditorPane.setPage(textFieldURL.getText());//显示 URL
        } catch (IOException ex) {
            JOptionPane msg = new JOptionPane();
            JOptionPane.showMessageDialog(this, "URL 地址不正确: "
                    + textFieldURL.getText(), "输入不正确! ", 0);
        }
    }

    void jEditorPane_hyperlinkUpdate(HyperlinkEvent e) { //响应页面打开超链接消息
        if (e.getEventType() ==
javax.swing.event.HyperlinkEvent.EventType.ACTIVATED) {
            try {
                URL url = e.getURL();                  //从消息中得到 URL
                jEditorPane.setPage(url);              //显示页面内容
                textFieldURL.setText(url.toString());  //显示 URL
            } catch (IOException io) {
                JOptionPane msg = new JOptionPane();
                JOptionPane.showMessageDialog(this,"打开该链接失败! ","输入不正确! ",
0);
            }
        }
    }

    protected void processWindowEvent(WindowEvent e) {    //处理窗体事件
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0); //关闭
        }
    }
    public static void main(String[] args) {     //Main 函数
        new WebBrowser();
    }
}
```

【运行结果】

运行结果如图 7-1 所示。

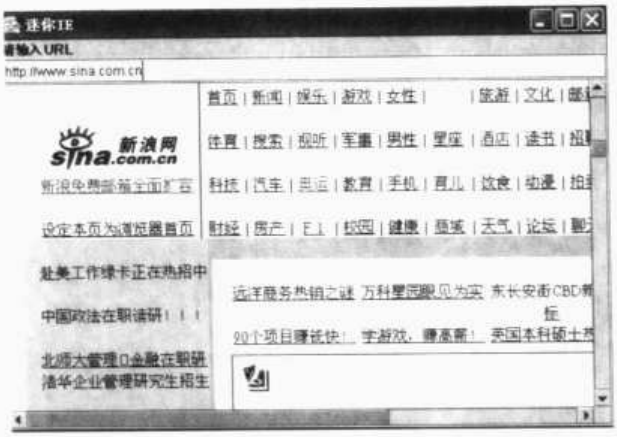


图 7-1

在文本框内输入网址并按回车键，稍等片刻就会得到网页的显示。

【分析与总结】

Swing 中的 HyperlinkEvent 事件表示超链接被按下时触发的，所以在该时间可以得到 URL，然后在 JEditorPane 中显示出链接的页面。

该程序只是简单的实现 IE 功能，还可给它添加更多的功能，例如：前进、后退、保存网页等等，编写出更加强大的 Web 浏览器来。

实例 80 Telnet

本实例实现远程登录（Telnet）功能。

【编程要点】

通过为通信双方建立 Socket 来传送消息，Socket 的构造函数有：

Socket()：根据 SocketImpl 的系统默认值生成一个没有连接的 socket。

Socket(InetAddress, int)：生成一个 socket 流并且把它连接到指定的 IP 地址的指定端口。

Socket(InetAddress, int, boolean)：生成一个 socket 并且把它连接到指定 IP 地址的指定端口。不推荐使用该方法。

Socket(InetAddress, int, InetAddress, int)：生成一个 socket 并且把它连接到指定远程端口的指定远程地址。

Socket(SocketImpl)：根据用户指定的 SocketImpl 生成一个无连接的 Socket。

Socket(String, int)：生成一个 socket 流并且把它连接到命名主机的指定端口。

Socket(String, int, boolean)：生成一个 socket 流并且把它连接到命名主机的指定端口。不推荐使用该方法。

Socket(String, int, InetAddress, int)：生成一个 socket 并且把它连接到在指定远程主机的指定远程端口。

【程序实现】

```
import java.io.*;
import java.net.*;

public class Telnet {
```

```
    public static void main(String [] args) throws IOException {
        if (args.length < 1) {
            System.out.println("用法: Telnet <主机名> <端口号>");
            return;
        }
        Socket sock = new Socket(args[0],
            (args.length < 2) ? 23 : Integer.parseInt(args[1]));
        //创建一个写线程
        new TelnetWriter(sock.getOutputStream()).start();
        //创建一个读线程
        new TelnetReader(sock.getInputStream()).start();
    }
}

class TelnetWriter extends Thread {
    private PrintStream out;
    public TelnetWriter(OutputStream out) {
        this.out = new PrintStream(out);
    }
    public void run() {
        try {
            //包装控制台输入流
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            //反复将控制台输入写到 Telnet 服务程序
            while (true)  out.println(in.readLine());
        } catch (IOException exc) {
            exc.printStackTrace();
        }
    }
}

class TelnetReader extends Thread {
    private InputStreamReader in;
    public TelnetReader(InputStream in) {
        this.in = new InputStreamReader(in);
    }
    public void run() {
        try {
            //反复将 Telnet 服务程序的反馈信息显示在控制台屏幕上
            while (true) {
                //从 Telnet 服务程序读取数据
                int b = in.read();
                if (b == -1)  System.exit(0);
                //将数据显示在控制台屏幕上
                System.out.print((char) b);
            }
        } catch (IOException exc) {
            exc.printStackTrace();
        }
    }
}
```

【运行结果】

在控制台输入 java telnet 主机名、端口号，该程序会反复将 Telnet 服务程序的反馈信息显示在控制台屏幕上。

【分析与总结】

telnet 建立的是可靠连接，因此可以看到反馈回的信息，知道远程登录是否成功。Socket 类可以为我们创建可靠连接。

实例 81　扫描应用端口

本实例实现扫描 TCP 端口。

【编程要点】

Socket类有个构造函数Socket(String host, int port)生成一个socket流并且把它连接到命名主机的指定端口。如果创建不成功，则抛出UnknownHostException或IOException异常。因此借此来判断port是否可用。

【程序实现】

```
import java.net.*;
import java.io.*;

public class PortScan {
    public static void main(String[] args) {
        String host = "localhost";
        if (args.length > 0) {
            host = args[0];
        }
        //扫描端口号小于1024的端口
        for (int i = 1; i < 1024; i++) {
            try {
                Socket s = new Socket(host, i);
                System.out.println("There is a server on port " + i + " of "
                        + host);
            } catch (UnknownHostException e) {
                System.err.println(e);
                break;
            } catch (IOException e) {
            }
        }
    }
}
```

【运行结果】

控制台输出（端口号使用情况因机器而异）：

```
There is a server on port 7 of localhost
There is a server on port 9 of localhost
There is a server on port 13 of localhost
There is a server on port 17 of localhost
There is a server on port 19 of localhost
There is a server on port 25 of localhost
```

实例82 聊天工具

本实例应用UDP协议的数据报文传输实现一个简单的聊天工具。

【编程要点】

与UDPSend的实例原理一样，再加上界面即可。

【程序实现】

```
import java.net.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.GregorianCalendar;
import javax.swing.JDialog;

public class QQ extends Frame implements ActionListener {
    Label label1 = new Label("请输入您要发送的信息（限英文）：");
    Label label2 = new Label("以下是你收到的消息记录：");
    Label label3 = new Label("把以上消息发给如下IP地址：");
    TextArea input = new TextArea("", 7, 14, TextArea.SCROLLBARS_BOTH);
    TextArea output = new TextArea("", 8, 14, TextArea.SCROLLBARS_BOTH);
    TextField IPAdd = new TextField("192.168.1.88");
    Button send = new Button("发送消息");
    Button about = new Button("关于");
```

```
Button clear = new Button("清空消息纪录");
GregorianCalendar time = new GregorianCalendar();
QQ() {
    super("仿 QQ 聊天工具");
    setLayout(null);
    setLocation(250, 250);
    this.setSize(518, 218);
    this.setResizable(false); //大小不可变
    this.setBackground(new Color(220, 220, 220));
    Toolkit kit = Toolkit.getDefaultToolkit();
    Image myImage = kit.getImage("QQ.bmp");
    this.setIconImage(myImage);

    label1.setFont(new Font("宋体", Font.PLAIN, 12));
    label1.setForeground(new Color(0, 0, 192));
    label1.setBounds(8, 28, 216, 16);

    input.setBackground(new Color(255, 255, 128));
    input.setFont(new Font("Times New Roman", Font.BOLD, 15));
    input.setForeground(Color.magenta);
    input.setBounds(8, 44, 248, 120);

    output.setBackground(new Color(128, 255, 255));
    output.setFont(new Font("Times New Roman", Font.PLAIN, 12));
    output.setForeground(Color.magenta);
    output.setBounds(264, 44, 248, 136);
    output.setEditable(false);

    send.setFont(new Font("新宋体", Font.PLAIN, 12));
    send.setLocation(136, 188);
    send.setSize(120, 22);

    clear.setFont(new Font("新宋体", Font.PLAIN, 12));
    clear.setLocation(392, 188);
    clear.setSize(120, 22);

    label2.setFont(new Font("宋体", Font.PLAIN, 12));
    label2.setForeground(new Color(0, 0, 192));
    label2.setBounds(264, 28, 216, 16);

    about.setFont(new Font("新宋体", Font.PLAIN, 12));
    about.setLocation(264, 188);
    about.setSize(120, 22);

    label3.setFont(new Font("宋体", Font.PLAIN, 12));
    label3.setForeground(new Color(0, 0, 192));
    label3.setBounds(8, 172, 160, 16);

    IPAdd.setFont(new Font("新宋体", Font.PLAIN, 12));
    IPAdd.setLocation(8, 190);
    IPAdd.setSize(120, 19);

    add(label1);
    add(input);
    add(label3);
    add(label2);
    add(output);
    add(IPAdd);
    add(send);
    add(about);
    add(clear);
    addWindowListener(new closeWin());
    send.addActionListener(this);
    about.addActionListener(this);
    clear.addActionListener(this);
    show();
    waitForData();
}
```

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == send)
            sendData();
        else if (e.getSource() == clear)
            output.setText("");
        else if (e.getSource() == about) {
            AboutQQ test = new AboutQQ(this);
        }
    }

    public static void main(String args[]) {
        new QQ();
    }

    void sendData() {
        try {
            String msg = input.getText();
            if (msg.equals(""))
                return;
            input.setText("");
            String ad = IPAdd.getText();
            InetAddress tea = InetAddress.getLocalHost();
            String asd = tea.getHostAddress();//发送方的IP地址
            output.append("[" + asd + "]:(" + time.get(GregorianCalendar.YEAR)
                    + "-" + time.get(GregorianCalendar.MONTH) + "-"
                    + time.get(GregorianCalendar.DATE) + " "
                    + time.get(GregorianCalendar.HOUR) + ":"
                    + time.get(GregorianCalendar.MINUTE) + ":"
                    + time.get(GregorianCalendar.SECOND) + ") " + "\n" + msg
                    + "\n");
            msg = "From [" + asd + "]:(" + time.get(GregorianCalendar.YEAR)
                    + "-" + time.get(GregorianCalendar.MONTH) + "-"
                    + time.get(GregorianCalendar.DATE) + " "
                    + time.get(GregorianCalendar.HOUR) + ":"
                    + time.get(GregorianCalendar.MINUTE) + ":"
                    + time.get(GregorianCalendar.SECOND) + ") \n" + msg;
            InetAddress address = InetAddress.getByName(ad);
            int len = msg.length();
            byte[] message = new byte[len];
            msg.getBytes(0, len, message, 0);
            DatagramPacket packet = new DatagramPacket(message, len, address,
                    9999);
            DatagramSocket socket = new DatagramSocket();
            socket.send(packet);
        } catch (Exception e) {
        }
    }

    void waitForData() {
        try {
            byte[] buffer = new byte[1024];
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
            DatagramSocket socket = new DatagramSocket(9999);
            while (true) {
                socket.receive(packet);
                String s = new String(buffer, 0, 0, packet.getLength());
                output.append(s + "\n");
                packet = new DatagramPacket(buffer, buffer.length);
            }
        } catch (Exception e) {
        }
    }
}

class closeWin extends WindowAdapter {
    public void windowClosing(WindowEvent e) {
        Frame fr = (Frame) (e.getSource());
        fr.dispose();
        System.exit(0);
    }
```

```
}

class AboutQQ {
    private Label label;
    private JDialog dialog;

    public AboutQQ(Frame f){
        label = new Label("Version 1.0");
        dialog = new JDialog(f, "About", true);
        dialog.setLocation(f.getLocation());
        Container dialogPane = dialog.getContentPane();
        dialogPane.setLayout(new BorderLayout());
        dialogPane.add(label);
        dialogPane.setBounds(50,50,50,50);
        dialog.pack();
        dialog.show();
    }
}
```

【运行结果】

运行结果如图 7-2 所示。

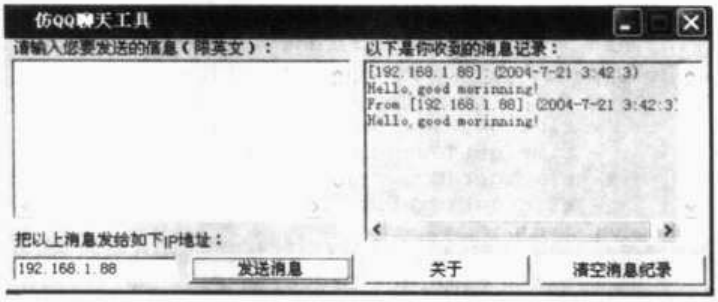


图 7-2

【分析与总结】

把握核心部分，不断扩展。

实例 83 上传文件

本实例演示如何实现上传文件的功能。

【编程要点】

Sockets 有两种主要的操作方式：面向连接的和无连接的。面向连接的 sockets 操作就像一部电话，它必须建立一个连接和一人呼叫。所有的事情在到达时的顺序与它们出发时的顺序时一样，无连接的 sockets 操作就像是一个邮件投递，没有什么保证，多个邮件可能在到达时的顺序与出发时的顺序不一样。

到底用哪种模式是由应用程序的需要决定的。如果可靠性更重要的话，用面向连接的操作会好一些。比如文件服务器需要它们的数据的正确性和有序性。如果一些数据丢失了，系统的有效性将会失去。

一些服务器，比如间歇性地发送一些数据块。如果数据丢了的话，服务器并不想要再重新发过一次。因为当数据到达的时候，它可能已经过时了。确保数据的有序性和正确性需要额外的操作的内存消耗，额外的费用将会降低系统的回应速率。因此，在聊天工具的设计时采用无连接的服务，而在上传文件时得采用面向连接的服务。

无连接的操作使用数据报协议，一个数据报是一个独立的单元，它包含了所有的这次投递的信息。把它想象成一个信封吧，它有目的地址和要发送的内容。这个模式下的 socket

不需要连接一个目的的 socket，它只是简单地投出数据报。无连接的操作是快速的和高效的，但是数据安全性不佳。

面向连接的操作使用 TCP 协议，一个在这个模式下的 socket 必须在发送数据之前与目的地的 socket 取得一个连接。一旦连接建立了，sockets 就可以使用一个流接口：打开-读-写-关闭。所有的发送的信息都会在另一端以同样的顺序被接收，面向连接的操作比无连接的操作效率更低，但是数据的安全性更高。

在 Java 中面向连接的类有两种形式，它们分别是客户端和服务器端。一个客户端向一个正在监听的服务器 socket 发出一个连接。客户端的 sockets 是用 Socket 类建立的服务器并不是主动地建立连接。

相反地，它们是被动地监听一个客户端的连接请示，然后给它们服务。服务器是由类 ServerSocket 来建立的。

下面的程序建立了一个服务器端 socket 并把它绑定到 80 端口：

```
ServerSocket serverSocket = new ServerSocket(80, 5);
```

第一个参数是服务器要监听的端口；第二个参数是可选的。API 文档中说明了这是一个监听时间，但是在传统的 socket 程序中第二个参数是监听深度。一个服务器可以同时接收多个连接请求，但是每次只能处理一个。监听堆是一个无回答的连接请求队列。上面的请求建立一个连接来处理最后五个请求。如果省略了后面的一个参数，则默认值是 50。一旦 socket 建立了并开始监听连接，进来的连接将会建立并放在监听堆。accetp()方法把在堆中的连接取出来。

```
Socket clientSocket = serverSocket.accept();
```

这个方法返回一个用来与来访者对话的客户端连接。服务器本身不可能建立对话，相反地，服务器 socket 会使用 accept()方法来产生一个新的 socket 服务器，socket 依旧打开并排列新的连接请求。

【程序实现】

```
//客户端 UploadClient.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;

public class UploadClient extends JFrame implements ActionListener {
    String ReadT2 = "";

    private JTextField[] T = new JTextField[4];
    private JButton ExitJB, OKJB, JOpen1, JOpen2, JOpen3;
    private JFileChooser choose = new JFileChooser();
    private File[] Myfile = new File[3];

    public UploadClient() {
        Container c = getContentPane();
        c.setLayout(new FlowLayout());

        T[0] = new JTextField(35);
        T[1] = new JTextField(35);
        T[2] = new JTextField(35);
        T[3] = new JTextField(45);
        ExitJB = new JButton("退出");
        OKJB = new JButton("上传");
        JOpen1 = new JButton("浏览...");
        JOpen2 = new JButton("浏览...");
```

```
        JOpen3 = new JButton("浏览...");
        ExitJB.addActionListener(this);
        OKJB.addActionListener(this);
        JOpen1.addActionListener(this);
        JOpen2.addActionListener(this);
        JOpen3.addActionListener(this);
        c.add(T[0]);
        c.add(JOpen1);
        c.add(T[1]);
        c.add(JOpen2);
        c.add(T[2]);
        c.add(JOpen3);
        c.add(OKJB);
        c.add(ExitJB);
        c.add(T[3]);
        T[0].setText("");
        T[1].setText("");
        T[2].setText("");
        T[3].setText("");
        setSize(500, 500);
        show();
    }

    public void actionPerformed(ActionEvent e) {
        ReadT2 = "";
        if (e.getSource() == ExitJB) {
            System.exit(0);
        }
        try {
            if (e.getSource() == OKJB) {
                for (int i = 0; i < 3; i++) {
                    if (T[i].getText() != "") {
                        this.client(i);
                    }
                }
            }
        } catch (Exception t) {
            System.out.println(t.toString());
        }

        if (e.getSource() == JOpen1) {
            this.openf(0);
        }
        if (e.getSource() == JOpen2) {
            this.openf(1);
        }
        if (e.getSource() == JOpen3) {
            this.openf(2);
        }
    }

    public void openf(int s) {
        int result = choose.showOpenDialog(null);
        Myfile[s] = choose.getSelectedFile();
        if (result == JFileChooser.APPROVE_OPTION) {
            T[s].setText(Myfile[s].getPath());
        } else if (result == JFileChooser.CANCEL_OPTION) {
            T[s].setText("");
            Myfile[s] = null;
        }
    }

    public void client(int s) {
        byte[] ff;
        int size;
        Socket clientSocket;
        PrintStream ps;
        String fileName;

        try {
```

```
                // 创建一个上传的 Socket
                clientSocket = new Socket("192.168.0.1", 8765);
                ps = new PrintStream(clientSocket.getOutputStream());
                size = (int) Myfile[s].length();
                fileName = Myfile[s].getName();
                BufferedReader br = new BufferedReader(new InputStreamReader(
                        clientSocket.getInputStream()));
                String over = "\n";
                byte pp[] = over.getBytes();
                byte byname[] = (fileName + over).getBytes();
                System.out.println(new String(byname));
                //文件名
                ps.write(byname);
                ps.flush();
                String gh = br.readLine();
                String tmp;
                tmp = String.valueOf(size) + over;
                byte bySize[] = tmp.getBytes();
                //文件大小
                ps.write(bySize);
                ps.flush();
                String jj = br.readLine();
                ReadT2 += "已上传的文件为" + gh.trim() + ";大小为" + jj.trim() + " ";
                T[3].setText(ReadT2);

                //文件内容
                FileInputStream in = new FileInputStream(Myfile[s]);
                BufferedInputStream bf = new BufferedInputStream(in);
                ff = new byte[1024];
                int c;
                while ((c = bf.read(ff)) != -1) {
                    ps.write(ff, 0, c);
                }

                ps.close();
                bf.close();
                in.close();
                clientSocket.close();
                T[s].setText("上传结束，请选择下个文件！");
                T[s].setText("");
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }

        public static void main(String args[]) {
            UploadClient app = new UploadClient();
            app.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        }
    }

    //服务器端 UploadServer.java
    import java.net.*;
    import java.io.*;
    import java.util.*;

    public class UploadServer {
        private static Socket clientSocket;
        private static int counter;

        public static void main(String[] args) {
            int i = 1;
            ServerSocket serverSocket;
            try {
                serverSocket = new ServerSocket(8765);
```

```
                while (true) {
                //监听与本 socket 的连接并且接受它
                    clientSocket = serverSocket.accept();
                    //创建线程，将上传数据写至服务器的文件中
                    Thread t = new MyThread(clientSocket, i);
                    t.start();
                    i++;
                }
            } catch (Exception e) {
                System.out.println(e);
            }
        }
}

//线程  MyThread.java
import java.net.*;
import java.io.*;

public class MyThread extends Thread {
    private Socket clientSocket;
    private int counter;

    public MyThread(Socket i, int c) {
        clientSocket = i;
        counter = c;
    }

    //采用同步机制
    public synchronized void run() {
        try {
            DataInputStream br = new DataInputStream(clientSocket
                    .getInputStream());
            DataOutputStream output = new DataOutputStream(clientSocket
                    .getOutputStream());
            //文件大小
            byte[] nn = new byte[225];
            br.read(nn);
            output.write(nn);
            output.flush();
            //文件名
            byte[] ll = new byte[255];
            br.read(ll);
            output.write(ll);
            output.flush();
            String tmpe = new String(ll);
            int aa = Integer.parseInt(tmpe.trim());
            //创建文件用于保存上传的文件
            String names = new String(nn);
            File MyFile = new File("d:\\");
            String files = "d:\\" + MyFile.separator + (names.trim());
            FileOutputStream out = new FileOutputStream(files);
            BufferedOutputStream bw = new BufferedOutputStream(out);
            byte ff[] = new byte[1024];
            int c = 0;
            //把上传的数据输出至文件
            while ((c = br.read(ff)) != -1) {
                bw.write(ff, 0, c);
            }
            bw.close();
            output.close();
            out.close();
            br.close();
            clientSocket.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

【运行结果】

客户机程序运行结果如图 7-3 所示。

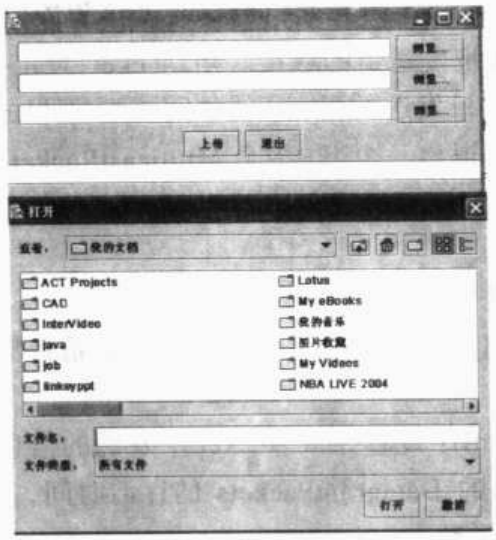


图 7-3

【分析与总结】

这里有两个程序：客户机程序和服务器程序，另外服务器程序还用到了读写线程类，把它独立为一个文件。注意 socket 流必须首先关闭，所有的 socket 流必须在 socket 关闭之前关闭。所有客户端程序都必须遵守下面的基本的步骤：

（1）建立客户端 socket 连接。

（2）得到 socket 的读和写的流。

（3）利用流。

（4）关闭流。

（5）关闭 socket。

所有的服务器都要有以下的基本的步骤：

（1）建立一个服务器 socket 并开始监听。

（2）使用 accept()方法取得新的连接。

（3）建立输入和输出流。

（4）在已有的协议上产生会话。

（5）关闭客户端流和 socket。

（6）回到第（2）步或者到第（7）步。

（7）关闭服务器 socket。

实例 84 IP 多组传播

本实例通过使用 IP 组播协议来实现在局域网组播组中发送和接受数据。

【编程要点】

MulticastSocket（多址发送报文端口(socket)类）继承自 DatagramSocket，用来发送和接收 IP 多址发送报文。MulticastSocket 是（UDP）报文端口，带有连接因特网上其他多址发送主机的附加功能。多址发送组由 D 类 IP 地址指定，范围在 224.0.0.1 到 239.255.255.255

之间，并包含一个标准 UDP 端口号码。要想连接--个多址发送组，可以首先创建一个带有所需端口的 MulticastSocket，再调用方法 joinGroup(InetAddress groupAddr)。给一个多址发送组发信息时，所有预订了该主机和端口的接收者接收该信息（在报文的存活时间内）。该 socket 不必是发信息的多址发送组的成员。当 socket 预定一个多址发送组/端口时，它会接收到其他主机向该组/端口发送的报文，就如同该组/端口的其他成员。socket 通过方法 leaveGroup(InetAddress addr)退出一个组。多个 MulticastSocket 可同时预定一个多址发送组和端口，并且它们都会接收到该组的报文。当前 Applet 不允许使用多址发送 socket。构造函数：

MulticastSocket()：创建一个多址发送 socket。

MulticastSocket(int)：创建多址发送 socket 并将其联系到特定端口。

主要方法：

public void setTTL(byte ttl)：设置在该 socket 上发送的信息包的缺省存活时间（TTL）。TTL 设置发送到多址发送组的 DatagramPackets 的存活时间，该时间定义了信息包死亡之前在网络上被转发的“hops”数。

joinGroup(InetAddress)：连接一个多址发送组。

leaveGroup(InetAddress)：脱离多址发送组。

public synchronized void send(DatagramPacket p, byte ttl)：将带有 TTL（存活时间）的分组报文包发送到目的地。此方法只在需要特定 TTL 的实例中使用；或者通常在该 socket 上设置一次 TTL，然后对所有信息包均使用此缺省 TTL。此方法不改变该 socket 的缺省 TTL。参数 p 为被发送的信息包。信息包必须包含目标多址发送 IP 地址以及被发送的数据。向目标多址发送地址发送信息包时不必是该组的成员。TTL 为多址发送信息包的可选存活时间。缺省 TTL 是 1。

【程序实现】

```
//package sendandrecievedata;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;

public class MulticastDemo {
    private boolean packFrame = false;
    //构建应用程序
    public MulticastDemo() {
        MulticastFrame frame = new MulticastFrame();
        if (packFrame) {
            frame.pack();
        } else {
            frame.validate();
        }
        //把窗口置于中心
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.width > screenSize.width) {
            frameSize.width = screenSize.width;
        }
        frame.setLocation((screenSize.width - frameSize.width) / 2,
                (screenSize.height - frameSize.height) / 2);
```

```
            frame.setVisible(true);
        }
        //Main 方法
        public static void main(String[] args) {
            try {
                UIManager.setLookAndFeel (UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {
                e.printStackTrace();
            }
            new MulticastDemo();
        }
    }
    class MulticastFrame extends JFrame {
        private JPanel contentPane;
        private TextField textField1 = new TextField();
        private Button button1 = new Button();
        private Label label1 = new Label();
        private Label label2 = new Label();
        int port; //声明组播使用的端口
        MulticastSocket socket;     //声明建立组播组使用的 MulticastSocket 类
        InetAddress group;          //声明建立组播组使用的组播组地址
        DatagramPacket packet;      //声明发送和接收数据所使用的 DatagramPacket 类
        //构造窗体
        public MulticastFrame() {
            enableEvents(AWTEvent.WINDOW_EVENT_MASK);
            try {
                init();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        //控件初始化
        private void init() throws Exception {
            contentPane = (JPanel) this.getContentPane(); //设计窗体布局
            textField1.setBounds(new Rectangle(88, 127, 240, 32));
            contentPane.setLayout(null);
            this.setSize(new Dimension(400, 300));
            this.setTitle("组播组中发送和接收数据");
            button1.setLabel("发送数据");
            button1.setBounds(new Rectangle(166, 186, 88, 29));
            button1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    button1_actionPerformed(e);
                }
            });
            label1.setBounds(new Rectangle(88, 84, 240, 32));
            label2.setBounds(new Rectangle(88, 41, 220, 32));
            contentPane.add(textField1, null);
            contentPane.add(button1, null);
            contentPane.add(label1, null);
            contentPane.add(label2, null);
            createMulticastGroupAndJoin();
        }
        //关闭窗体时退出
        protected void processWindowEvent(WindowEvent e) {
            super.processWindowEvent(e);
            if (e.getID() == WindowEvent.WINDOW_CLOSING) {
                System.exit(0);
            }
        }
        public void createMulticastGroupAndJoin()   //创建一个组播组并加入此组的函数
        {
            try {
                port = 5000;                        //设置组播组的监听端口为 5000
                group = InetAddress.getByName("239.0.0.0"); //设置组播组的地址为
239.0.0.0
```

```
                socket = new MulticastSocket(port);          //初始化 MulticastSocket 类
并将端口号与之关联
                socket.setTimeToLive(1);                     //设置组播数据报的发送范围为本地网络
                socket.setSoTimeout(10000);                  //设置套接字的接收数据报的最长时间
                socket.joinGroup(group);                     //加入此组播组
                label2.setText("已加入地址为 239.0.0.0 的组播组");
            } catch (Exception e1) {
                System.out.println("Error: " + e1);//捕捉异常情况
            }
        }
        public void sendData() //向组播组发送数据的函数
        {
            try
            {
                byte[] data = textField1.getText().getBytes();   //将用户要发送的数据转
换为字节形式并存储在数组中
                packet = new DatagramPacket(data, data.length, group, port); //初始化
DatagramPacket
                socket.send(packet);        //通过 MulticastSocket 实例端口向组播组发送数据
            } catch (Exception e1) {
                System.out.println("Error: " + e1);          //捕捉异常情况
            }
        }
        public String recieveData()          //从组播组接收数据的函数
        {
            String message;
            try
            {
                packet.setData(new byte[512]);               //设定接收数据的
DatagramPacket 实例的数组大小
                packet.setLength(512);      //设定接收数据的 DatagramPacket 实例的长度
                socket.receive(packet);     //通过 MulticastSocket 实例端口从组播组接收数据
                //将接收的数据转换成字符串形式
                message = new String(packet.getData());
            } catch (Exception e1) {
                System.out.println("Error: " + e1);      //捕捉异常情况
                message = "Error: " + e1;
            }
            return message;
        }
        //按钮 1 事件处理
        void button1_actionPerformed(ActionEvent e) {
            sendData(); //首先向组播组发送数据
            String message = recieveData();              //然后从组播组接受此数据
            label1.setText("Data recieved: '" + message + "'");  //本机显示接收到的数据
        }
    }
```

【运行结果】

运行结果如图 7-4 所示。



图 7-4

【分析与总结】

本实例采用DatagramPacket类实例作为数据传输的载体，并使用MulticastSocket对象发送和接受数据，实际上是UDP传送的扩展。

实例85 邮件发送

本程序演示如何发送电子邮件。

【编程要点】

电子邮件传递可以由多种协议来实现。目前，在Internet网上最流行的三种电子邮件协议是SMTP、POP3和IMAP。简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）是一个运行在TCP/IP之上的协议，用它发送和接收电子邮件。SMTP服务器在默认端口25上监听。SMTP客户使用一组简单的、基于文本的命令与SMTP服务器进行通信。在建立了一个连接后，为了接收响应，SMTP客户首先发出一个命令来标识它们的电子邮件地址。如果SMTP服务器接受了发送者发出的文本命令，它就利用一个OK响应和整数代码确认每一个命令。客户发送的另一个命令意味着电子邮件消息体的开始，消息体以一个圆点“.”加上回车符终止。

SMTP协议是目前网上流行的发送E-Mail的协议，SMTP协议共有14条命令。不过，发一封E-Mail只需用如下5条命令就足够了，分别为：

HELO < SP > < domain > < CRLF >，与SMTP服务器握手，传送本机域名。

MAIL < SP > FROM：< reverse-path > < CRLF >，传送发信者的信箱名称。

RCPT < SP > TO：< forward-path > < CRLF >，传送接收者的信箱名称。

DATA < CRLF >，发送信件数据（包括信头和信体）。

QUIT < CRLF >，退出与SMTP服务器的连接。

【程序实现】

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.net.*;
import java.io.*;
import javax.swing.*;

public class MailTest {
    public static void main(String[] args) {
        JFrame frame = new MailTestFrame();
        frame.show();
    }
}
class MailTestFrame extends JFrame implements ActionListener {
    private BufferedReader in;
    private PrintWriter out;
    private JTextField from;
    private JTextField to;
    private JTextField smtpServer;
    private JTextArea message;
    private JTextArea response;
    public MailTestFrame() {
        setTitle("MailTest");
        setSize(300, 300);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
```

```
        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        //当容器比组件要大时，只调整垂直方向组件的大小
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 0;
        gbc.weighty = 0;
        gbc.weightx = 0;
        add(new JLabel("发送地址:"), gbc, 0, 0, 1, 1);
        gbc.weightx = 100;
        from = new JTextField(20);
        add(from, gbc, 1, 0, 1, 1);
        gbc.weightx = 0;
        add(new JLabel("收件地址:"), gbc, 0, 1, 1, 1);
        gbc.weightx = 100;
        to = new JTextField(20);
        add(to, gbc, 1, 1, 1, 1);
        gbc.weightx = 0;
        add(new JLabel("SMTP 服务器:"), gbc, 0, 2, 1, 1);
        gbc.weightx = 100;
        smtpServer = new JTextField(20);
        add(smtpServer, gbc, 1, 2, 1, 1);
        gbc.fill = GridBagConstraints.BOTH;
        gbc.weighty = 100;
        message = new JTextArea();
        add(new JScrollPane(message), gbc, 0, 3, 2, 1);
        response = new JTextArea();
        add(new JScrollPane(response), gbc, 0, 4, 2, 1);
        gbc.weighty = 0;
        JButton sendButton = new JButton("发送");
        sendButton.addActionListener(this);
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(sendButton);
        add(buttonPanel, gbc, 0, 5, 2, 1);
    }
    private void add(Component c, GridBagConstraints gbc, int x, int y, int w,
            int h) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }
    public void actionPerformed(ActionEvent evt) {
        //异步机制，使得当所有的 awt 事件发生后才进行以下工作
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                sendMail();
            }
        });
    }
    public void sendMail() {
        try {
            Socket s = new Socket(smtpServer.getText(), 25);
            out = new PrintWriter(s.getOutputStream());
            in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            String hostName = InetAddress.getLocalHost().getHostName();
            send(null);
            //传送本机域名
            send("HELO " + hostName);
            //传送发信者信箱名称
            send("MAIL FROM: " + from.getText());
            //传送收信者信箱名称
            send("RCPT TO: " + to.getText());
            //发送信箱数据，包括信头和信体
            send("DATA");
            out.println(message.getText());
            send(".");
            s.close();
```

```
            } catch (IOException exception) {
                response.append("Error: " + exception);
            }
        }
        public void send(String s) throws IOException {
            if (s != null) {
                response.append(s + "\n");
                out.println(s);
                out.flush();
            }
            String line;
            if ((line = in.readLine()) != null)
                response.append(line + "\n");
        }
}
```

【运行结果】

运行结果如图 7-5 所示。



图 7-5

【分析与总结】

在邮件发送程序中，依然是按照下面的顺序进行：

（1）建立客户端 socket 连接。

（2）利用流，发送 SMTP 的命令和邮件内容。

（3）关闭流。

（4）关闭 socket。

实例 86 FTP 浏览器

本实例实现一个 FTP（文件传输协议）浏览器。

【编程要点】

利用 Socket 建立与目标服务器的连接，注意端口的设置问题。

【程序实现】

```
import java.net.*;
import java.io.*;
import java.util.*;

class FtpClient {
    static final boolean debug = false;
    public static final int FTP_PORT = 21;    //ftp 的端口
    static int FTP_SUCCESS = 1;
    static int FTP_TRY_AGAIN = 2;
    static int FTP_ERROR = 3;
    //数据传输套接字
    private Socket  dataSocket = null;
    private boolean replyPending = false;
    private boolean binaryMode = false;
```

```
        private boolean passiveMode = false;
        String user = null;                          //登录用的用户名
        String password = null;                      //登录用的密码
        String command;                              //最近一次命令
        int lastReplyCode;                           //最近的响应信息
        public String welcomeMsg;                    //服务器的欢迎信息
        //从服务器返回的响应字符串对象
        protected Vector serverResponse = new Vector(1);
        protected Socket serverSocket = null;        //与服务器通信的套接字
        public PrintWriter  serverOutput;
        public InputStream  serverInput;             //从服务器读取响应的缓冲流
        /** 返回服务器连接的状态 */
        public boolean serverIsOpen() {
            return serverSocket != null;
        }
         /** 设置为被动模式 */
         public void setPassive(boolean mode) {
             passiveMode = mode;
         }
         /** 读取服务器的响应信息 */
        public int readServerResponse() throws IOException {
            StringBuffer replyBuf = new StringBuffer(32);
            int c;
            int continuingCode = -1;
            int code = -1;
            String response;
            if (debug) System.out.println("readServerResponse start");
            try{
            while (true) {
                if (debug) System.out.println("readServerResponse outer while loop: "+
serverInput.available());
                while ((c = serverInput.read()) != -1) {
                   if (c == '\r') {
                       if ((c = serverInput.read()) != '\n')
                           replyBuf.append('\r');
                    }
                    replyBuf.append((char)c);
                    if (c == '\n')
                       break;
                }
                if (debug) System.out.println("Now past inner while loop");
                response = replyBuf.toString();
                replyBuf.setLength(0);
                if (debug) {
                    System.out.print(response);
                }
                try {
                    code = Integer.parseInt(response.substring(0, 3));
                } catch (NumberFormatException e) {
                    code = -1;
                } catch (StringIndexOutOfBoundsException e) {
                    //此行不存在响应码，循环跳到下一次
                    continue;
                }
                serverResponse.addElement(response);
                if (continuingCode != -1) {
                    /* we've seen a XXX- sequence */
                    if (code != continuingCode ||
                        (response.length() >= 4 && response.charAt(3) == '-')) {
                        continue;
                    } else {
                        continuingCode = -1;         //到达程序的结尾
                        break;
                    }
                } else if (response.length() >= 4 && response.charAt(3) == '-') {
                    continuingCode = code;
                    continue;
```

```
            } else {
                break;
            }
        }
        }catch(Exception e){e.printStackTrace();}
        if (debug) System.out.println("readServerResponse done");
        return lastReplyCode = code;
    }
    /** 发送命令 <i>cmd</i> 给服务器 */
    public void sendServer(String cmd) {
        if (debug) System.out.println("sendServer start");
        serverOutput.println(cmd);
        if (debug) System.out.println("sendServer done");
    }
    /** 返回服务器的所有响应字符串 */
    public String getResponseString() {
        String s = new String();
        for(int i = 0;i < serverResponse.size();i++) {
          s+=serverResponse.elementAt(i);
        }
        serverResponse = new Vector(1);
        return s;
    }
    /** 获取响应字符串 */
    public String getResponseStringNoReset() {
        String s = new String();
        for(int i = 0;i < serverResponse.size();i++) {
          s+=serverResponse.elementAt(i);
        }
        return s;
    }
    /** 发送 QUIT 命令给服务器并关闭连接 */
    public void closeServer() throws IOException {
        if (serverIsOpen()) {
            issueCommand("QUIT");
        if (! serverIsOpen()) {
             return;
            }
            serverSocket.close();
            serverSocket = null;
            serverInput = null;
            serverOutput = null;
        }
    }
    protected int issueCommand(String cmd) throws IOException {
        command = cmd;
        int reply;
        if (debug) System.out.println(cmd);
        if (replyPending) {
            if (debug) System.out.println("replyPending");
            if (readReply() == FTP_ERROR)
                System.out.print("Error reading pending reply\n");
        }
        replyPending = false;
        do {
            sendServer(cmd);
            reply = readReply();
            if (debug) System.out.println("in while loop of issueCommand method");
        } while (reply == FTP_TRY_AGAIN);
        return reply;
    }
    //检测命令
    protected void issueCommandCheck(String cmd) throws IOException {
        if (debug) System.out.println("issueCommandCheck");
        if (issueCommand(cmd) != FTP_SUCCESS) {
            throw new FtpProtocolException(cmd);
            }
    }
    /** 读取返回数据 */
```

```
protected int readReply() throws IOException {
    lastReplyCode = readServerResponse();
    switch (lastReplyCode / 100) {
    case 1:
        replyPending = true;
    case 2://这个 case 用来以后扩展功能
    case 3:
        return FTP_SUCCESS;
    case 5:
        if (lastReplyCode == 530) {
            if (user == null) {
                throw new FtpLoginException("Not logged in");
            }
            return FTP_ERROR;
        }
        if (lastReplyCode == 550) {
            if (!command.startsWith("PASS"))
                throw new FileNotFoundException(command);
            else
                throw new FtpLoginException("Error: Wrong Password!");
        }
    }
    return FTP_ERROR;
}
// 打开数据连接
protected Socket openDataConnection(String cmd) throws IOException {
    ServerSocket portSocket = null;
    String portCmd;
    InetAddress myAddress = InetAddress.getLocalHost();
    byte addr[] = myAddress.getAddress();
    int shift;
    String ipaddress;
    int port = 0;
    IOException e;
    if (this.passiveMode) {
        //首先尝试被动模式传输
        try {
            getResponseString();
            if (issueCommand("PASV") == FTP_ERROR) {
                e = new FtpProtocolException("PASV");
                throw e;
            }
            String reply = getResponseStringNoReset();
            reply = reply.substring(reply.lastIndexOf("(") + 1, reply
                    .lastIndexOf(")"));
            StringTokenizer st = new StringTokenizer(reply, ",");
            String[] nums = new String[6];
            int i = 0;
            while (st.hasMoreElements()) {
                try {
                    nums[i] = st.nextToken();
                    i++;
                } catch (Exception a) {
                    a.printStackTrace();
                }
            }
            ipaddress = nums[0] + "." + nums[1] + "." + nums[2] + "."
                    + nums[3];
            try {
                int firstbits = Integer.parseInt(nums[4]) << 8;
                int lastbits = Integer.parseInt(nums[5]);
                port = firstbits + lastbits;
            } catch (Exception b) {
                b.printStackTrace();
            }
            if ((ipaddress != null) && (port != 0)) {
                dataSocket = new Socket(ipaddress, port);
            } else {
                e = new FtpProtocolException("PASV");
                throw e;
```

```
                }
                if (issueCommand(cmd) == FTP_ERROR) {
                    e = new FtpProtocolException(cmd);
                    throw e;
                }
            } catch (FtpProtocolException fpe) {
                portCmd = "PORT ";
                //附加 host 地址
                for (int i = 0; i < addr.length; i++) {
                    portCmd = portCmd + (addr[i] & 0xFF) + ",";
                }
                try {
                    portSocket = new ServerSocket(20000);
                    //附加端口
                    portCmd = portCmd
                            + ((portSocket.getLocalPort() >>> 8) & 0xff) + ","
                            + (portSocket.getLocalPort() & 0xff);
                    if (issueCommand(portCmd) == FTP_ERROR) {
                        e = new FtpProtocolException("PORT");
                        throw e;
                    }
                    if (issueCommand(cmd) == FTP_ERROR) {
                        e = new FtpProtocolException(cmd);
                        throw e;
                    }
                    dataSocket = portSocket.accept();
                } finally {
                    if (portSocket != null)
                        portSocket.close();
                }
                dataSocket = portSocket.accept();
                portSocket.close();
            }
        }
        else { //端口传送
            portCmd = "PORT ";
            //附加 host 地址
            for (int i = 0; i < addr.length; i++) {
                portCmd = portCmd + (addr[i] & 0xFF) + ",";
            }
            try {
                portSocket = new ServerSocket(20000);
                //附加端口号
                portCmd = portCmd + ((portSocket.getLocalPort() >>> 8) & 0xff)
                        + "," + (portSocket.getLocalPort() & 0xff);
                if (issueCommand(portCmd) == FTP_ERROR) {
                    e = new FtpProtocolException("PORT");
                    throw e;
                }
                if (issueCommand(cmd) == FTP_ERROR) {
                    e = new FtpProtocolException(cmd);
                    throw e;
                }
                dataSocket = portSocket.accept();
            } finally {
                if (portSocket != null)
                    portSocket.close();
            }
            dataSocket = portSocket.accept();
            portSocket.close();
        }
        return dataSocket;
    }
    /** 打开一个到 <i>host </i>的 FTP 连接 */
    public void openServer(String host) throws IOException,
                                        UnknownHostException {
        int port = FTP_PORT;
        if (serverSocket != null)
            closeServer();
```

```
            serverSocket = new Socket(host, FTP_PORT);
            serverOutput = new PrintWriter(new
BufferedOutputStream(serverSocket.getOutputStream()),true);
            serverInput = new BufferedInputStream(serverSocket.getInputStream());
        }
        /** 打开到 host <i>host </i> 端口为 <i>port </i>的 FTP 连接 */
        public void openServer(String host, int port) throws IOException,
UnknownHostException {
            if (serverSocket != null)
               closeServer();
            serverSocket = new Socket(host, port);
            serverOutput = new PrintWriter(new
BufferedOutputStream(serverSocket.getOutputStream()),
                                    true);
            serverInput = new BufferedInputStream(serverSocket.getInputStream());
            if (readReply() == FTP_ERROR)
               throw new FtpConnectException("Welcome message");
        }
        /** 使用用户名 <i>user</i> 和密码 <i>password</i> 登录*/
        public void login(String user, String password) throws IOException {
            if (!serverIsOpen())
               throw new FtpLoginException("Error: not connected to host.\n");
            this.user = user;
            this.password = password;
            if (issueCommand("USER " + user) == FTP_ERROR)
               throw new FtpLoginException("Error: User not found.\n");
            if (password != null && issueCommand("PASS " + password) == FTP_ERROR)
               throw new FtpLoginException("Error: Wrong Password.\n");
        }
        /** 只用用户名 <i>user</i> 不用密码登录 */
        public void login(String user) throws IOException {
            if (!serverIsOpen())
               throw new FtpLoginException("not connected to host");
            this.user = user;
            if (issueCommand("USER " + user) == FTP_ERROR)
               throw new FtpLoginException("Error: Invalid Username.\n");
        }
        /** 以 Ascii 模式从 FTP server 获取一个文件 */
        public BufferedReader getAscii(String filename) throws IOException {
            Socket  s = null;
            try {
               s = openDataConnection("RETR " + filename);
            }           catch           (FileNotFoundException           fileException)
{fileException.printStackTrace();}
            return new BufferedReader( new InputStreamReader(s.getInputStream()));
        }
        /** 以 Binary 模式从 FTP server 获取一个文件 */
        public BufferedInputStream getBinary(String filename) throws IOException {
            Socket  s = null;
            try {
               s = openDataConnection("RETR " + filename);
            }           catch           (FileNotFoundException           fileException)
{fileException.printStackTrace();}
            return new BufferedInputStream(s.getInputStream());
        }
        /** 以 Ascii 模式发送一个文件 */
        public BufferedWriter putAscii(String filename) throws IOException {
            Socket s = openDataConnection("STOR " + filename);
            return new BufferedWriter(new OutputStreamWriter(s.getOutputStream()),4096);
        }
        /** 以 Binary 模式向 server 发送一个文件 */
        public BufferedOutputStream putBinary(String filename) throws IOException {
            Socket s = openDataConnection("STOR " + filename);
            return new BufferedOutputStream(s.getOutputStream());
        }
        /** 以 Ascii 模式在 server 上创建或添加一个文件 */
        public BufferedWriter appendAscii(String filename) throws IOException {
            Socket s = openDataConnection("APPE " + filename);
```

```
        return new BufferedWriter(new OutputStreamWriter(s.getOutputStream()),4096);
    }
    /** 以 Binary 模式在 server 上创建或添加一个文件 */
    public BufferedOutputStream appendBinary(String filename) throws IOException {
        Socket s = openDataConnection("APPE " + filename);
        return new BufferedOutputStream(s.getOutputStream());
    }
    /** NLIST 文件在远端 FTP server */
    public BufferedReader nlist() throws IOException {
        Socket s = openDataConnection("NLST");
        return new BufferedReader( new InputStreamReader(s.getInputStream()));
    }
    /** LIST files 在远端 FTP server */
    public BufferedReader list() throws IOException {
        Socket s = openDataConnection("LIST");
        return new BufferedReader( new InputStreamReader(s.getInputStream()));
    }
    /** 在 FTP server 上改变文件路径 */
    public void cd(String remoteDirectory) throws IOException {
        issueCommandCheck("CWD " + remoteDirectory);
    }
    /** 在 server 上改变文件名 */
    public void rename(String oldFile, String newFile) throws IOException {
         issueCommandCheck("RNFR " + oldFile);
         issueCommandCheck("RNTO " + newFile);
    }
    /** Site 命令 */
    public void site(String params) throws IOException {
         issueCommandCheck("SITE "+ params);
    }
    /** 设置为'I'传输模式 */
    public void binary() throws IOException {
        issueCommandCheck("TYPE I");
        binaryMode = true;
    }
    /** 设置为'A'传输模式 */
    public void ascii() throws IOException {
        issueCommandCheck("TYPE A");
        binaryMode = false;
    }
    /** 发送 Abort 命令 */
    public void abort() throws IOException {
        issueCommandCheck("ABOR");
    }
    /** 在远端系统浏览上一级目录 */
    public void cdup() throws IOException {
        issueCommandCheck("CDUP");
    }
    /** 在远端系统中创建一个目录 */
    public void mkdir(String s) throws IOException {
        issueCommandCheck("MKD " + s);
    }
    /** 在远端系统中删除某个路径 */
    public void rmdir(String s) throws IOException {
        issueCommandCheck("RMD " + s);
    }
    /** 删除文件 */
    public void delete(String s) throws IOException {
        issueCommandCheck("DELE " + s);
    }
    /** 获取当前目录名 */
    public void pwd() throws IOException {
        issueCommandCheck("PWD");
    }
    /** 获取远端系统的信息 */
    public void syst() throws IOException {
```

```
        issueCommandCheck("SYST");
    }
    /** 新的 FTP 客户端连接到 host <i>host</i>. */
    public FtpClient(String host) throws IOException {
        openServer(host, FTP_PORT);
    }
    /** 新的 FTP 客户端连接到 host <i>host</i>, port <i>port</i>. */
    public FtpClient(String host, int port) throws IOException {
        openServer(host, port);
    }
    /** 演示 */
    public static void main (String args []) throws IOException{
      System.out.println("Demo of FtpClient class.\n");
      //标准登录过程
      FtpClient f = new FtpClient("ftp.zsu.edu.cn");
      System.out.print(f.getResponseString());
      f.login("anonymous");
      System.out.print(f.getResponseString());
      f.pwd();
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      f.setPassive(true);

      //使用列表
      System.out.println("\nDemo of nlist() function");
      f.ascii();  //把客户端设为 ASCII 模式以获取文本列表
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      BufferedReader t = f.nlist();     //f.list 提供了更多的一些细节信息
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      while( true ) {
        String stringBuffer = t.readLine();
        if( stringBuffer == null ) break;
        else System.out.println(stringBuffer);
      }
      t.close();
      System.out.print(f.getResponseString());
      //下面使用 getAscii() 函数获取文件; 函数 getBinary() 也类似
      System.out.println("\nDemo of getAscii() function");
      f.ascii();       // 设置传输模式为 ASCII
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      BufferedReader bufGet = f.getAscii("welcome.msg");
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      PrintWriter     pOut     =     new     PrintWriter(new     BufferedWriter(new
FileWriter("welcome.msg")));
      int i;
      char c[] = new char[4096];
      while ((i = bufGet.read(c)) != -1)
        pOut.write(c,0,i);
      bufGet.close();
      pOut.close();
      System.out.print(f.getResponseString());
      //下面使用 appendAscii()函数添加一个文件, appendBinary()函数的使用类似
      System.out.println("\nDemo of appendAscii() function");
      BufferedWriter bufAppe;
      String localFile = "file name goes here";
      f.ascii();
      System.out.println(f.command);
      try {
      bufAppe = f.appendAscii(localFile);
      System.out.println(f.command);
      System.out.print(f.getResponseString());
      FileReader fIn = new FileReader(localFile);
      BufferedReader bufIn = new BufferedReader(fIn);
      int k;
```

```
            char b[] = new char[1024];
            while ((k = bufIn.read(b)) != -1)
              bufAppe.write(b,0,k);
            bufIn.close();
            bufAppe.flush();
            bufAppe.close();
            }catch(Exception appendErr) {
               System.out.println(appendErr.toString());//printStackTrace();

            }
            System.out.print(f.getResponseString());
            //下面使用putBianary()函数发送一个二进制文件，函数putAscii()类似
            System.out.println("\nDemo of putBinary() function");
            String localFile2 = "file name goes here";
            f.binary();
            System.out.println(f.command);
            BufferedOutputStream bufPut = f.putBinary(localFile2);
            System.out.println(f.command);
            System.out.print(f.getResponseString());
            BufferedInputStream bufIn = new BufferedInputStream(new
FileInputStream(localFile2));
            int j;
            byte b[] = new byte[1024];
            while ((j = bufIn.read(b)) != -1)
              bufPut.write(b,0,j);
            bufIn.close();
            bufPut.flush();
            bufPut.close();
            System.out.print(f.getResponseString());
            //关闭连接
            f.closeServer();
            System.out.println(f.command);
            System.out.print(f.getResponseString());
          }
      }
      class FtpLoginException extends FtpProtocolException {
         FtpLoginException(String s) { super(s); }
      }
      class FtpConnectException extends FtpProtocolException {
         FtpConnectException(String s) { super(s); }
      }
      class FtpProtocolException extends IOException {
         FtpProtocolException(String s) { super(s); }
      }
```

【运行结果】

根据提示操作连接一个预先设定的FTP站点测试功能，实现FTP客户端连接服务器及上下传等基本功能。

【分析与总结】

在FtpClient类中定义了多个方法实现多种功能，如支持被动模式等。在代码中有许多形如：

```
if (debug) System.out.println("readServerResponse start");
```

这样的语句，是用于调试提示信息的输出，debug这个布尔变量用来控制提示信息是否输出。

小结

网络API对Java语言有着举足轻重的作用，利用Java网络API编写的网络程序比其他语言要容易得多。通过这些API，可以非常容易地在Internet上发送或接收数据，浏览Web页面、发送或接受电子邮件等。

第 8 章　Java 数据库编程

本章通过 7 个实例介绍了 Java 的数据库编程，其中包括 JDBC-ODBC 桥、数据库常见操作、使用 JDBC 驱动程序、显示数据库的内容、数据库查询和修改、事务使用、设置保存点等。数据库编程与其他 Java 技术结合紧密，比如 JSP、EJB 等。

实例 87　JDBC-ODBC 桥

本实例使用 JDBC-ODBC 连接 Microsoft Access 数据库。

【编程要点】

Java 语言是编写数据库应用程序的杰出语言之一，它提供了方便访问数据的技术。利用 Java 语言中的 JDBC 技术，用户能方便地开发出基于 Web 网页的数据库访问程序，从而扩充网络应用功能。

JDBC(Java Database Connectivity，Java 数据库连接)是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一的访问接口。

JDBC 由一组用 Java 语言编写的类与接口组成，通过调用这些类和接口所提供的方法，用户能够以一致的方式连接多种不同的数据库系统（如 Access、SQL Server 2000、Oracle、Sybase 等），进而可使用标准的 SQL 语言来存取数据库中的数据，而不必再为每一种数据库系统编写不同的 Java 程序代码。

JDBC-ODBC 桥将 JDBC 数据转换为 ODBC 数据来源，然后利用 ODBC 与数据库连接。下面是设置数据库连接的步骤：

（1）在 Access 中建立一个名为 student 的数据库，数据库文件命名为 student.mdb，表名是 STUDENTS，其中属性有文本类型 ID，文本类型 NAME，数字类型 AGE，ID 为主键。

（2）在“控制面板”中的“管理工具”中配置 ODBC 数据源，打开“ODBC 数据源管理器”的“系统 DSN”选项。单击“添加”按钮，显示“创建新数据源”对话框，选择“Microsoft Access Driver(*.mdb)”选项，单击“完成”按钮，显示“ODBC Microsoft Access 安装”对话框，在“数据源名”文本框中输入“student”，单击“选择”按钮显示“选择数据源”对话框，然后在本地文件系统中选择数据库文件，单击“确定”按钮。

【实例程序】

```
import java.sql.*;
public class JDBCDemo
{
    private String dbURL;           //数据库标识名
    private String user;            //数据库用户
    private String password;        //数据库用户密码
    public static void main(String args[])
    {
        try
        {
            JDBCDemo bridge=new JDBCDemo();
            bridge.setURL("jdbc:odbc:student");
            bridge.setUser("");
            bridge.setPassword("");
            Connection con=bridge.getConnection();  //得到数据库连接
```

```
            System.out.println(con.getCatalog());    //打印当前数据库目录名称
            con.close();
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
    public Connection getConnection()
    {
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    //装载数据库驱动
            return DriverManager.getConnection(dbURL,user,password);
        }
        catch (Exception e)
        {
            System.out.println(e.toString());
        }
        return null;
    }
    public void setURL(String dbURL)
    {
        this.dbURL=dbURL;              //设置数据库标识
    }
    public void setUser(String user)
    {
        this.user=user;                //设置当前用户
    }
    public void setPassword(String password)
    {
        this.password=password;        //设置用户密码
    }
}
```

【运行结果】

若数据库连接成功，控制台将输出该数据库文件的绝对路径；若数据库连接失败，则输出异常的信息处理：

```
java.sql.SQLException:[Microsoft][ODBC 驱动程序 管理器]
java.lang.NULLPointerException
```

第一个异常表示未发现数据源名称并且未指定默认驱动程序。第二个异常是 DSN 名设置或驱动程序错误。

JDBC-ODBC 桥使 ODBC 驱动程序可被用作 JDBC 驱动程序。它的实现为 JDBC 的快速发展提供了一条途径，其长远目标提供一种访问某些不常见的 DBMS（如果对这些不常见的 DBMS 未实现 JDBC）的方法。

实例 88　数据库的常见操作

本实例沿用上一例的数据库，演示如何利用 Statement 操作数据库，可事先给数据库添加数据，用于演示删除、添加等的操作。

【编程要点】

JDBC API 最重要的接口如下：

（1）java.sql.DriverManager：处理驱动的调入并且对产生新的数据库连接提供支持。

（2）java.sql.Connection：代表对特定数据库的连接。

（3）java.sql.Statement：代表一个特定的容器来对一个特定的数据库执行 SQL 语句。

（4）java.sql.ResultSet：控制对一个特定语句的行数据的存取。其中 java.sql.Statement 又有两个子类型：

① java.sql.PreparedStatement：用于执行预编译的 SQL 语句。

② java.sql.CallableStatement：用于执行对一个数据库内嵌过程的调用。

通过 Statement 类的 executeQuery(String command)来实现如查询、修改、添加和删除等的操作。

【程序实现】

```
import java.sql.*;
public class DBOperation
{
    private Statement sta;          //用于数据库操作的 Statement
    public static void main(String args[])
    {
        try
        {
            JDBCODBCDemo bridge=new JDBCODBCDemo();
            DBOperation operator=new DBOperation();
            ResultSet rs;               //结果集
            String sqlCommand;          //用于操作的 SQL 命令
            bridge.setURL("jdbc:odbc:student");
            bridge.setUser("");
            bridge.setPassword("");
            Connection con=bridge.getConnection();
            operator.setStatement(con);         //设置 Satement
            //查询操作
            sqlCommand="SELECT * FROM STUDENTS";
            System.out.println("\n########## Query ##########");
            System.out.println("input SQL command"+sqlCommand+"\n");
            rs=operator.executeQuery(sqlCommand);
            rs.next();
            for(int i=0;i<rs.getRow();i++)
            {
                System.out.println(rs.getString("NAME"));
                rs.next();
            }
            //修改操作
            sqlCommand="UPDATE STUDENTS SET AGE=22 WHERE NAME=\'Andy\'";
            System.out.println("\n########## modify ##########");
            System.out.println("input SQL command"+sqlCommand+"\n");
            operator.executeUpdate(sqlCommand);
            sqlCommand="SELECT * FROM STUDENTS WHERE NAME=\'Andy\'";
            rs=operator.executeQuery(sqlCommand);
            rs.next();
            for(int i=0;i<rs.getRow();i++)
            {
                System.out.println(rs.getString("NAME")+
                    "'s age is now "+rs.getInt("AGE"));
                rs.next();
            }
            //插入操作
            sqlCommand="INSERT INTO STUDENTS(ID,NAME,AGE)"+
                "VALUES(\'5\',\'Lilei\',1)";
            System.out.println("\n########## Add ##########");
            System.out.println("??SQL??"+sqlCommand+"\n");
            operator.executeInsert(sqlCommand);
            sqlCommand="SELECT * FROM STUDENTS WHERE ID=\'5\'";
            rs=operator.executeQuery(sqlCommand);
            rs.next();
            for(int i=0;i<rs.getRow();i++)
            {
                System.out.println("ID "+rs.getString("ID")+
                    " added "+rs.getString("NAME"));
                rs.next();
```

```
            }
            //删除操作
            sqlCommand="DELETE FROM STUDENTS WHERE ID=\'5\'";
            System.out.println("\n########## Delete ##########");
            System.out.println("input SQL command "+sqlCommand+"\n");
            operator.executeDelete(sqlCommand);
            sqlCommand="SELECT * FROM STUDENTS WHERE ID=\'5\'";
            rs=operator.executeQuery(sqlCommand);
            rs.next();
            if(!rs.next())
            {
                System.out.println("ID = 5 doesn't exists");
            }
            //??????,????
            rs.close();
            operator.closeStatement();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
    public void setStatement(Connection con)
    {
        try
        {
            this.sta=con.createStatement();     //Statement 中有关数据库查询的函数
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }

    public ResultSet executeQuery(String sqlCommand)
    {
        try
        {
            //Statement 中修改数据库的函数
            return sta.executeQuery(sqlCommand);
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
        return null;
    }

    public void executeUpdate(String sqlCommand)
    {
        try
        {
            sta.executeUpdate(sqlCommand);
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
    public void executeInsert(String sqlCommand)
    {
        try
        {
            sta.executeUpdate(sqlCommand);
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
```

```
    }
    public void executeDelete(String sqlCommand)
    {
        try
        {
            sta.executeUpdate(sqlCommand);
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
    public void closeStatement()
    {
        try
        {
            sta.close();

        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
}
```

【运行结果】

在控制台可以看到表的添加、修改和删除的结果。

【分析与总结】

本实例采用的 ResultSet 类对象，数据库修改、添加和删除操作使用 excuteUpdate(String command)方法，查询操作使用 executeQuery(String command)方法并返回一 ResultSet 对象。其中 next()方法自当前记录向下移动一行。一个 ResultSet 游标在初始化时位于第一个数据前，所以使用前需要调用 next()方法使其移动到第一个记录行。最后必须关闭所有的连接，并释放所有的资源。

实例 89 使用 JDBC 驱动程序连接 SQL Server

本实例演示如何利用数据库供应方提供的 JDBC 驱动程序连接数据库。

【编程要点】

JDBC 驱动程序总共有四种类型：

（1）JDBC-ODBC 桥，再加上 ODBC 驱动程序。

（2）本机 API，部分是 Java 的驱动程序。

（3）面向数据库中间件的纯 Java 驱动程序。

（4）直接面向数据库的纯 Java 驱动程序。

第（3）、（4）两类都是纯 Java 的驱动程序，因此，对于 Java 开发者来说，它们在性能、可移植性、功能等方面都有优势。前面介绍了如何使用第一类的驱动程序，这里将用第四类的驱动程序。

多种数据库都有自己的 JDBC 驱动程序，SQL Server 的 JDBC 驱动程序可以在微软的网站上免费获得，安装后，将该安装目录下的 lib 目录下的 jar 文件加入系统的环境变量 CLASSPATH 中，或者把它添加到 Java 编译工具的 library 中。

本例中先在 SQLServer 建立一个 student 数据库。数据库访问的用户名和密码都为 sa。

【程序实现】

```
//JDBCDemo.java
//Connect database with JDBC 2.0 Driver

import java.sql.*;
public class JDBCDemo
{
    private String dbURL;           //数据库标识名
    private String user;            //数据库用户
    private String password;        //数据库用户密码

    public Connection getConnection()
    {
        try
        {
            //装载数据库驱动
            Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
            return DriverManager.getConnection(dbURL,user,password);
        }
        catch (Exception e)
        {
            System.out.println(e.toString());
        }
        return null;
    }
    public void setURL(String dbURL)
    {
        this.dbURL=dbURL;           //设置数据库标识
    }
    public void setUser(String user)
    {
        this.user=user;             //设置当前用户
    }
    public void setPassword(String password)
    {
        this.password=password;     //设置用户密码
    }

    public static void main(String args[])
    {
        try
        {
            JDBCDemo driver=new JDBCDemo();
            driver.setURL("jdbc:microsoft:sqlserver://192.168.28.129:1433;
DatabaseName=student");
            driver.setUser("sa");
            driver.setPassword("sa");
            Connection con=driver.getConnection();  //得到数据库连接
            System.out.println(con.getCatalog());   //打印当前数据库目录名称
            con.close();
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
    }
}
```

【运行结果】

在控制台将输出数据库的名称。

【分析与总结】

本实例通过 Class.ForClass("com.microsoft.jdbc.sqlserver.SQLServerDriver")加载 MS SQLServer 的 JDBC 驱动程序，然后通过设置数据库的路径 driver.setURL("jdbc:microsoft:sqlserver://192.168.28.129:1433;DatabaseName=student") 来获

得数据库的连接，其中//192.168.28.129 是数据库服务器的 IP，1433 是端口，student 是数据库名。

实例 90 显示数据库的内容

本实例将数据库的信息在 Swing 组件上显示出来。

【编程要点】

这里用到了 DatabaseMetaData 类，该类提供了关于数据库的整体信息。它的许多方法将在 ResultSets 中返回信息列表。可以使用一般的 ResultSet()方法(如 getString()和 getInt()) 从这些 ResultSet()中检索数据。如果给定的元数据格式是不可用的，这些方法将抛出一个 SQLException。

Properties 类描述了一个持久的特性集。Properties 可被保存在流中或从流中装入。特性列表中的每个关键字和它的相应值是一个字符串。一个特性列表可包含另一个特性列表作为它的“缺省”；如果特性关键字没有在原始的特性列表中找到，则搜索第二个特性列表。本实例将数据库连接的信息都存储在 ViewDB.properpties 文件中。Properties 对象可以从该文件读入信息。这里用到的是上一例出现过的 student 数据库。

【程序实现】

```
//ViewDB.java

import java.net.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
public class ViewDB {
    public static void main(String[] args) {
        JFrame frame = new ViewDBFrame();
        frame.show();
    }
}

class ViewDBFrame extends JFrame implements ActionListener {
    private JButton nextButton;
    private JPanel dataPanel;
    private JComboBox tableNames;
    private ArrayList fields;
    private Connection con;
    private Statement stmt;
    private DatabaseMetaData md;
    private ResultSet rs;

    public ViewDBFrame()
    {
        setTitle("ViewDB");
        setSize(300, 200);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        Container contentPane = getContentPane();
        //组合框，用来显示表名
        tableNames = new JComboBox();
        tableNames.addActionListener(this);
```

```
        dataPanel = new JPanel();
        contentPane.add(dataPanel, "Center");
        nextButton = new JButton("Next");
        nextButton.addActionListener(this);
        JPanel p = new JPanel();
        p.add(nextButton);
        contentPane.add(p, "South");
        fields = new ArrayList();

        try {
            con = getConnection();
            stmt = con.createStatement();
            md = con.getMetaData();          //得到数据库的整体信息
            //获得可用于一个分类表中的表的说明
            ResultSet mrs = md.getTables(null, null, null,
                new String[] { "TABLE" }); //获得可用于一个分类表中的表的说明
            while (mrs.next())
                tableNames.addItem(mrs.getString(3));
            mrs.close();
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, e);
        }
        contentPane.add(tableNames, "North");
    }

    public static Connection getConnection() throws SQLException, IOException {
        Properties props = new Properties();
        String fileName = "ViewDB.properties";
        FileInputStream in = new FileInputStream(fileName);
        //从输入流读特性列表。
        props.load(in);
      //用该特性列表中的指定关键字 jdbc.drivers 来查找特性。
        String drivers = props.getProperty("jdbc.drivers");
        if (drivers != null)  //设置 Properties 参数指示的系统属性。
            System.setProperty("jdbc.drivers", drivers);
        String url = props.getProperty("jdbc.url");
        String username = props.getProperty("jdbc.username");
        String password = props.getProperty("jdbc.password");

        return DriverManager.getConnection(url, username, password);
    }

    private void add(Container p, Component c, GridBagConstraints gbc, int x,
            int y, int w, int h) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        p.add(c, gbc);
    }

    public void actionPerformed(ActionEvent evt) {
        if (evt.getSource() == nextButton) {
            showNextRow();          //显示下一个表项
        }
        else if (evt.getSource() == tableNames) {
            //如果是组合框的选项发生了变化
            remove(dataPanel);
            dataPanel = new JPanel();
            fields.clear();
            dataPanel.setLayout(new GridBagLayout());
            GridBagConstraints gbc = new GridBagConstraints();
            gbc.weighty = 100;

            try {
                String tableName = (String) tableNames.getSelectedItem();
                if (rs != null)
                    rs.close();
```

```
                rs = stmt.executeQuery("SELECT * FROM " + tableName);
                //ResultSetMetaData 对象可以用于查找 ResultSet 中的列的类型和特性
                ResultSetMetaData rsmd = rs.getMetaData();
                for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                    //获得打印输出和显示的列标题
                    String columnName = rsmd.getColumnLabel(i);
                    //获得列的正常的最大字符宽度
                    int columnWidth = rsmd.getColumnDisplaySize(i);
                    JTextField tb = new JTextField(columnWidth);
                    fields.add(tb);

                    gbc.weightx = 0;  //指定如何分布额外的水平空间为 0
                    gbc.anchor = GridBagConstraints.EAST;
                    gbc.fill = GridBagConstraints.NONE;
                    add(dataPanel, new JLabel(columnName), gbc, 0, i - 1, 1, 1);

                    gbc.weightx = 100;
                    gbc.anchor = GridBagConstraints.WEST;
                    //设置为在水平方向上改变组件的大小，而不是垂直方向
                    gbc.fill = GridBagConstraints.HORIZONTAL;
                    add(dataPanel, tb, gbc, 1, i - 1, 1, 1);
                }
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this, e);
            }
            getContentPane().add(dataPanel, "Center");
            doLayout();
            pack();
            showNextRow();
        }
    }

    public void showNextRow() {
        if (rs == null)
            return;
        {
            try {
                if (rs.next()) {
                    for (int i = 1; i <= fields.size(); i++) {
                        //把当前行的列值作为一个 Java String 获取
                        String field = rs.getString(i);
                        JTextField tb = (JTextField) fields.get(i - 1);
                        tb.setText(field);
                    }
                } else {
                    rs.close();
                    rs = null;
                }
            } catch (Exception e) {
                System.out.println("Error " + e);
            }
        }
    }
}
```

【运行结果】

运行结果如图 8-1 所示。



图 8-1

【分析与总结】

本例从 Connection 对象中调用 getMetaData()方法获得数据库的整体信息 DatabaseMetaData，然后调用 getTables()获得数据库的表名。从查询数据库的结果集 ResultSet 对象中获得 ResultSetMetaData 对象，并用它的 getColumnLabel()方法获得表的列名，和通过 getString(i)获得每个表项的第 i 列的值。

实例91 数据库查询和修改

本实例演示数据库的查询和修改，并通过 Swing 组件表现出来。

【编程要点】

本实例在查询和更新操作时将一条 SQL 语句是预编译的且存放在 PreparedStatement 对象中。该对象可用于有效地多次执行该语句。

本程序用到的数据库 book，包含 Authors、Books、Publishers、BooksAuthors 表，其中 Authors 包含有 Author_ID、Name、URL；Books 包含有 Title(书名)、ISBN、Publisher_ID、URL、Price 等列属性；BooksAuthors 包含有 ISBN、Author_ID、Seq_no（该作者在该书是第几作者）等；Publishers 包含有 Publisher_ID、Name、URL。

【程序实现】

```
//QueryDB.java
import java.net.*;
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;

public class QueryDB {
    public static void main(String[] args) {
        JFrame frame = new QueryDBFrame();
        frame.show();
    }
}

class QueryDBFrame extends JFrame implements ActionListener {
    private JComboBox authors;
    private JComboBox publishers;
    private JTextField priceChange;
    private JTextArea result;
    private Connection con;
    private Statement stmt;
    private PreparedStatement authorQueryStmt;
    private PreparedStatement authorPublisherQueryStmt;
    private PreparedStatement publisherQueryStmt;
    private PreparedStatement allQueryStmt;

    public QueryDBFrame() {
        setTitle("QueryDB");
        setSize(400, 300);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        getContentPane().setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();

        authors = new JComboBox();
```

```
        authors.setEditable(false);
        authors.addItem("Any");

        publishers = new JComboBox();
        publishers.setEditable(false);
        publishers.addItem("Any");

        result = new JTextArea(4, 50);
        result.setEditable(false);

        priceChange = new JTextField(8);
        priceChange.setText("-5.00");

        try {
            //连接数据库
            con = getConnection();
            stmt = con.createStatement();

            //将数据库中的作者名添加到组合框
            String query = "SELECT Name FROM Authors";
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next())
                authors.addItem(rs.getString(1));

            //将出版社名添加到组合框
            query = "SELECT Name FROM Publishers";
            rs = stmt.executeQuery(query);
            while (rs.next())
                publishers.addItem(rs.getString(1));
        } catch (Exception e) {
            result.setText("Error " + e);
        }

        gbc.fill = GridBagConstraints.NONE;
        gbc.weightx = 100;
        gbc.weighty = 100;
        add(authors, gbc, 0, 0, 2, 1);

        add(publishers, gbc, 2, 0, 2, 1);

        gbc.fill = GridBagConstraints.NONE;
        JButton queryButton = new JButton("Query");
        queryButton.addActionListener(this);
        add(queryButton, gbc, 0, 1, 1, 1);

        JButton changeButton = new JButton("Change prices");
        changeButton.addActionListener(this);
        add(changeButton, gbc, 2, 1, 1, 1);

        gbc.fill = GridBagConstraints.HORIZONTAL;
        add(priceChange, gbc, 3, 1, 1, 1);

        gbc.fill = GridBagConstraints.BOTH;
        add(result, gbc, 0, 2, 4, 1);
    }

    /** 利用属性文件中的信息连接数据库 **/
    public static Connection getConnection() throws SQLException, IOException {
        Properties props = new Properties();
        String fileName = "QueryDB.properties";
        FileInputStream in = new FileInputStream(fileName);
        props.load(in);

        String drivers = props.getProperty("jdbc.drivers");
        if (drivers != null)
            System.setProperty("jdbc.drivers", drivers);
        String url = props.getProperty("jdbc.url");
        String username = props.getProperty("jdbc.username");
        String password = props.getProperty("jdbc.password");
```

```
        return DriverManager.getConnection(url, username, password);
    }

    private void add(Component c, GridBagConstraints gbc, int x, int y, int w,
            int h) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
        gbc.gridheight = h;
        getContentPane().add(c, gbc);
    }

    public void actionPerformed(ActionEvent evt) {
        String arg = evt.getActionCommand();
        if (arg.equals("Query")) { //用户按下 Query 按钮
            ResultSet rs = null;
            try {
                String author = (String) authors.getSelectedItem();
                String publisher = (String) publishers.getSelectedItem();
                if (!author.equals("Any") && !publisher.equals("Any")) {
                    if (authorPublisherQueryStmt == null) {
                    //根据用户选择的出版社名和作者名查询相关的书名和书价
                    String authorPublisherQuery ="SELECT Books.Price,Books.Title"
                        + "FROM Books, BooksAuthors, Authors, Publishers "
                        + "WHERE Authors.Author_Id = BooksAuthors.Author_Id AND "
                        + "BooksAuthors.ISBN = Books.ISBN AND "
                        + "Books.Publisher_Id = Publishers.Publisher_Id AND "
                        + "Authors.Name = ? AND "
                        + "Publishers.Name = ?";
                        authorPublisherQueryStmt = con
                                .prepareStatement(authorPublisherQuery);
                    }

                    authorPublisherQueryStmt.setString(1, author);
                    authorPublisherQueryStmt.setString(2, publisher);
                    rs = authorPublisherQueryStmt.executeQuery();
                } else if (!author.equals("Any") && publisher.equals("Any")) {
                    if (authorQueryStmt == null) {
                        //根据用户选择的作者名查询相关的书名和书价
                        String authorQuery = "SELECT Books.Price, Books.Title "
                            +"FROM Books,BooksAuthors,Authors "
                            +"WHERE Authors.Author_Id=BooksAuthors.Author_Id AND"
                            +"BooksAuthors.ISBN = Books.ISBN AND "
                            +"Authors.Name = ?";
                        authorQueryStmt = con.prepareStatement(authorQuery);
                    }
                    authorQueryStmt.setString(1, author);
                    rs = authorQueryStmt.executeQuery();
                } else if (author.equals("Any") && !publisher.equals("Any")) {
                    if (publisherQueryStmt == null) {
                        //根据用户选择的出版社名查询相关的书名和书价
                        String publisherQuery = "SELECT Books.Price, Books.Title "
                        +"FROM Books, Publishers "
                        +"WHERE Books.Publisher_Id = Publishers.Publisher_Id AND "
                        + "Publishers.Name = ?";
                         publisherQueryStmt = con
                              .prepareStatement(publisherQuery);
                    }
                    publisherQueryStmt.setString(1, publisher);
                    rs = publisherQueryStmt.executeQuery();
                } else {
                    if (allQueryStmt == null) {
                        //若用户未选任何信息，则输出所有的书名和对应的书价
                        String allQuery = "SELECT Books.Price, Books.Title FROM
Books";
                        allQueryStmt = con.prepareStatement(allQuery);
                    }
                    rs = allQueryStmt.executeQuery();
                }
```

```
                result.setText("");
                while (rs.next())
                    result.append(rs.getString(1) + "|" + rs.getString(2) + "\n");
                rs.close();
            } catch (Exception e) {
                result.setText("Error " + e);
            }
        } else if (arg.equals("Change prices")) { //用户选择 "Change prices" 按钮
            String publisher = (String) publishers.getSelectedItem();
            if (publisher.equals("Any"))
                result.setText("I am sorry, but I cannot do that.");
            else
                try {
                    //根据用户输入的新的书价更新 Books 表的数据
                    String updateStatement = "UPDATE Books "
                        + "SET Price = Price + "
                        + priceChange.getText()
                        + " WHERE Books.Publisher_Id = "
                        + "(SELECT Publisher_Id FROM Publishers WHERE Name = '"
                        + publisher + "')";
                    int r = stmt.executeUpdate(updateStatement);
                    result.setText(r + " records updated.");
                } catch (Exception e) {
                    result.setText("Error " + e);
                }
        }
    }

    public void dispose() {
        try {
            stmt.close();
            con.close();
        } catch (SQLException e) {
        }
    }
}
```

【运行结果】

运行结果如图 8-2 所示。

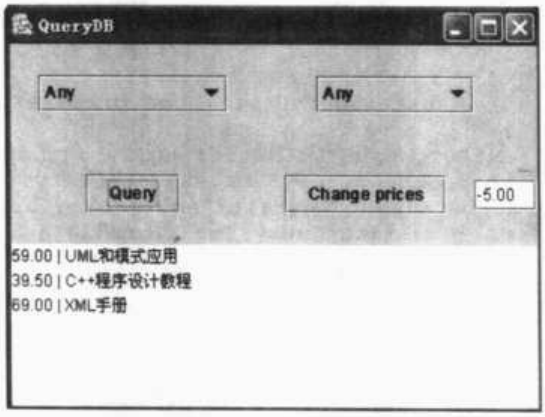


图 8-2

【分析与总结】

本例在要获得用户输入的信息后再进行数据库的查询和修改工作，通过 PerparedStatement 的 setString(int, String)方法来实现，参考下面程序：

```
String publisherQuery = "SELECT Books.Price, Books.Title "
                + "FROM Books, Publishers "
                + "WHERE Books.Publisher_Id = Publishers.Publisher_Id AND "
                + "Publishers.Name = ?";
publisherQueryStmt = con.prepareStatement(publisherQuery);
publisherQueryStmt.setString(1, publisher);
```

实例92 事务使用

本实例演示如何使用事务来进行数据库操作。

【编程要点】

在数据库操作中，一个事务是由一个或是多个表达式组成的一个不可分割的工作单元。

在JDBC中，可以通过提交commit()或回滚rollback()结束事务的操作，默认为自动提交，也可通过调用setAutoCommit(false)禁止自动提交。禁止后即可把多个数据库操作的表达式作为一个事务，在操作完成后调用commit()整体提交。如果其中一个表达式操作失败，则不执行到commit()且产生响应的异常，此时可在异常捕获时调用rollback()返回。于是便可保持多次更新操作后，相关数据的一致性。

本程序用到的是一个Access数据库coffee.mdb，它包含一个表COFFEES，含有属性COF_NAME（文本类型）、SALE（数字类型）、TOTAL（数字类型）。其中COF_NAME表示咖啡的名字，SALE表示本周的咖啡销量，TOTAL为咖啡总销量。

【程序实现】

```
//TransactionDemo.java
import java.sql.*;

public class TransactionDemo {
    public static void main(String args[]) {
        String url = "jdbc:odbc:coffee";
        Connection con = null;
        Statement stmt;
        PreparedStatement updateSales;
        PreparedStatement updateTotal;
        String updateString = "update COFFEES " +
                              "set SALES = ? where COF_NAME = ?";
        String updateStatement = "update COFFEES " +
                 "set TOTAL = TOTAL + ? where COF_NAME = ?";
        String query = "select COF_NAME, SALES, TOTAL from COFFEES";
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
        catch(java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }
        try {
            con = DriverManager.getConnection(url,"", "");
            updateSales = con.prepareStatement(updateString);
            updateTotal = con.prepareStatement(updateStatement);
            int [] salesForWeek = {175, 150, 60, 155, 90};
            String [] coffees = {"Colombian", "French_Roast",
                                 "Espresso", "Colombian_Decaf",
                                 "French_Roast_Decaf"};
            int len = coffees.length;
            //禁止自动提交，设置回滚点
            con.setAutoCommit(false);
            for (int i = 0; i < len; i++) {
                updateSales.setInt(1, salesForWeek[i]);
                updateSales.setString(2, coffees[i]);
                updateSales.executeUpdate();
                updateTotal.setInt(1, salesForWeek[i]);
                updateTotal.setString(2, coffees[i]);
                updateTotal.executeUpdate();
                con.commit();  //事务提交
            }
            con.setAutoCommit(true);
            updateSales.close();
```

```
                updateTotal.close();
                stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(query);
                while (rs.next()) { //打印 COFFEES 表的信息
                    String c = rs.getString("COF_NAME");
                    int s = rs.getInt("SALES");
                    int t = rs.getInt("TOTAL");
                    System.out.println(c + "     " + s + "    " + t);
                }
                stmt.close();
                con.close();
            } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
                if (con != null) {
                    try {
                        System.err.print("Transaction is being ");
                        System.err.println("rolled back");
                        con.rollback(); //操作不成功，回滚
                    } catch(SQLException excep) {
                        System.err.print("SQLException: ");
                        System.err.println(excep.getMessage());
                    }
                }
            }
        }
    }
```

【运行结果】

控制台输出更新后的 COFFEES 表信息，或者是 Transaction is being rolled back 的信息。

【分析与总结】

JDBC API 提供了五种操作支持事务，允许或禁止脏数据读写（dirty reads）、重复读写（repeatable reads）和影像读写（phantomreads）三种操作。JDBC 根据数据库提供的默认值设置事务支持及其加锁，也允许手工设置并可使用 getTransactionIsolation()查看数据库的当前设置。注意在手动设置时，数据库及其驱动程序必须支持相应的事务操作。

实例 93 设置保存点

本实例演示在数据库事务处理中设置保存点。

【编程要点】

也许在 JDBC 3.0 中最令人兴奋的附加特点就是 Savepoint 了。JDBC 2 中的事务支持让开发人员可以控制对数据的并发访问，从而保证持续数据总是保持一致的状态。可惜的是，有时候需要的是对事务多一点的控制，而不是在当前的事务中简单地对每一个改变进行回滚。

在 JDBC 3.0 下，可以通过 Savepoint 获得这种控制。Savepoint 接口允许将事务分割为各个逻辑断点，以控制有多少事务需要回滚。

图 8-3 将说明如何在事务中运用 Savepoint。

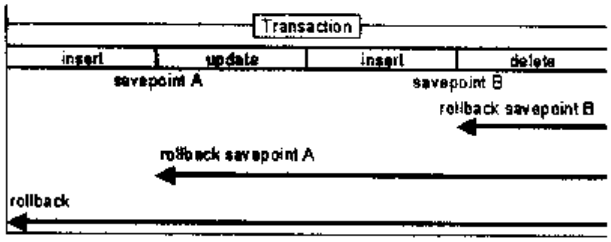


图 8-3

保存点是事务中的一个特殊记号，它允许将那些在它建立后执行的命令全部回滚，把事务的状态恢复到保存点所在的时刻。在 JDBC 中可以通过 Connection 对象的 setSavepoint() 方法来设置保存点，然后在有必要恢复时再调用 Connection 对象的 rollback()方法将保存点恢复。这里用到的数据库仍然是上例中的 coffee 数据库。

【程序实现】

```
//SetSavepoint.java

import java.sql.*;

public class SetSavepoint {

    public static void main(String args[]) {
        String url = "jdbc:microsoft:sqlserver://192.168.28.129:1433;
        DatabaseName=coffee;SelectMethod=cursor";
        try {
            Class.forName(com.microsoft.jdbc.sqlserver.SQLServerDriver");
        }
        catch (java.lang.ClassNotFoundException e) {
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }

        try {
            Connection con = DriverManager.getConnection(url, "","");
            con.setAutoCommit(false);
            String query = "SELECT COF_NAME, PRICE FROM COFFEES "
                    + "WHERE TOTAL > ?";
            String update = "UPDATE COFFEES SET PRICE = ? "
                    + "WHERE COF_NAME = ?";
            PreparedStatement getPrice = con.prepareStatement(query);
            PreparedStatement updatePrice = con.prepareStatement(update);
            getPrice.setInt(1, 7000);
            ResultSet rs = getPrice.executeQuery();
            //设置保存点
            Savepoint save1 = con.setSavepoint();
            while (rs.next()) {
                String cof = rs.getString("COF_NAME");
                float oldPrice = rs.getFloat("PRICE");
                float newPrice = oldPrice + (oldPrice * .05f);
                updatePrice.setFloat(1, newPrice);
                updatePrice.setString(2, cof);
                updatePrice.executeUpdate();
                System.out.println("New price of " + cof + " is " + newPrice);
                if (newPrice > 11.99) {
                    //在一定条件下回滚
                    con.rollback(save1);
                }
            }
            getPrice = con.prepareStatement(query);
            updatePrice = con.prepareStatement(update);
            getPrice.setInt(1, 8000);
            rs = getPrice.executeQuery();
            System.out.println();
            Savepoint save2 = con.setSavepoint();
            while (rs.next()) {
                String cof = rs.getString("COF_NAME");
                float oldPrice = rs.getFloat("PRICE");
                float newPrice = oldPrice + (oldPrice * .05f);
                updatePrice.setFloat(1, newPrice);
                updatePrice.setString(2, cof);
                updatePrice.executeUpdate();
                System.out.println("New price of " + cof + " is " + newPrice);
                if (newPrice > 11.99) {
                    con.rollback(save2);
                }
            }
```

```
                con.commit();
                Statement stmt = con.createStatement();
                rs = stmt.executeQuery("SELECT COF_NAME, " + "PRICE FROM COFFEES");
                System.out.println();
                while (rs.next()) {
                    String name = rs.getString("COF_NAME");
                    float price = rs.getFloat("PRICE");
                    System.out.println("Current price of " + name + " is " + price);
                }
                con.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

【运行结果】

运行后可以看到事务处理后的表的信息。

【分析与总结】

控制事务的回滚操作带来更大的方便。

小结

JDBC API 为 Java 应用程序提供了连接数据库、获取数据库信息、执行 SQL 命令并处理其产生的结果以及对数据库中的数据进行修改和更新等操作的标准方法。本章的实例都是 JDBC 的基本应用，通过使用 JDBC API，可以开发出功能强大的数据库应用程序。

第 9 章　Java B/S 结构编程

本章通过 14 个实例介绍了 JSP、Servlet 等技术。其中：包括读写文件、网站计数器、汉字处理、文件上传、获取客户机浏览器信息、收发邮件、JSP 连接数据库、用 JavaBean 连接数据库、网络定餐、在 Servlet 中实时地创建图像、简单购物车、用户注册登录系统。

实例 94　HelloWorld

本实例比较简单，在下载了应用服务器（如 tomcat5.0）后在 IE 上运行该程序显示几行不同字体大小的文本。

【编程要点】

JSP 主要是由一些 JSP 标记再加上一些 Java 代码组成。运行 JSP 程序需要额外的 Web 服务器，这里用 tomcat，下载 tomcat 服务器安装在 C:\Tomcat 5.0 目录下。

【实例程序】

```
<%@ page language="java" %>
<%@ page contentType="text/html; charset=GB2312"%>
<html>
<head>
<title>Hello World</title>
<meta http-equiv="Content-Type" content="content=text/html; charset=gb2312">
</head>
<body bgcolor="#FFFFFF">
<center>
<font size= 5 color=blue>多种字体大小显示</font>
<center>
<br>
<hr>
<br>
<div align="center">
<%
     //控制文字大小的循环语句
     for( int i=1; i<=6; i++ )
       out.println( "<h" + i + ">Hello World!</h" + i + ">" );
%>
</div>
</body>
</html>
```

【运行结果】

在 C:\Tomcat 5.0\webapps\ROOT 下保存 HelloWorld.jsp 文件。运行服务器并打开 IE，输入 http://localhost:8080/HelloWorld.jsp（这里假设安装 tomcat 时设置的端口为 8080），运行结果如图 9-1 所示。（注：本机 IP 为 127.0.0.1）



图 9-1

【分析与总结】

程序中开头两行用于设置 JSP 页面的属性（脚本语言，字符集），通过<% %>标记可以在 JSP 页面中添加 Java 代码。

实例 95 读写文件

本实例演示如何在 JSP 中读写文件。

【编程要点】

利用 FileReader 进行读文件，PrintWriter 进行写文件。

【实例程序】

ReadWrite.jsp 的文件内容如下：

```
<%@ page language="java" import="java.io.**%>
<%@ page contentType="text/html;charset=gb2312"%>
<html>
<head>
<title>读写文件</title>
<body>
<center>
<font size=6 color=blue>
读写文件示例
</font>
</center>
<%

//读文件
//变量声明
String strFileName;                    //文件名
File objFile;                          //文件对象
FileReader objFileReader;              //读文件对象
char[] chrBuffer = new char[10];       //缓冲
int intLength;                         //实际读出的字符数(一个中文为一个字符)
//设置待读文件名
strFileName = "d:\\test.txt";
//创建文件对象
objFile = new File(strFileName);
//判断文件是否存在
if(objFile.exists()){//文件存在
  //创建读文件对象
  objFileReader = new FileReader(objFile);
  //读文件内容
  while((intLength=objFileReader.read(chrBuffer))!=-1){
    //输出
    out.write(chrBuffer,0,intLength);
  }
  //关闭读文件对象
  objFileReader.close();
}
else{//文件不存在
  out.println("下列文件不存在: "+strFileName);
}
%>
<%
//写文件
String str = "预祝 2008 年奥运成功! ";
//always give the path from root. This way it almost always works.
String nameOfTextFile = "D:\\out.txt";
try {
```

```
    PrintWriter pw = new PrintWriter(new FileOutputStream(nameOfTextFile));
    pw.println(str);
    //clean up
    pw.close();
  } catch(IOException e) {
    out.println(e.getMessage());
  }
  %>
  </body>
  </head>
  </html>
```

【运行结果】

在 C:\Tomcat 5.0\webapps\ROOT 下保存 ReadWrite.jsp 文件，在 D 盘创建 test.txt 文件。运行服务器并打开 IE，输入 http://localhost:8080/ReadWrite.jsp，运行结果如图 9-2 所示，test.txt 文件中的内容在浏览器上显示出来，而在 D 盘根目录下多了一个文件 out.txt 是该程序的输出内容。（注：本机 IP 为 127.0.0.1）



图 9-2

实例 96　网站计数器

本实例实现一个简单的网站计数器。

【编程要点】

这里主要用到了 Java 2 的同步处理机制，每个 Java application 都有 basic.counter 的属性，当 application 被调用一次就把 basic.counter 值加 1。

【程序实现】

```
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page language="java" %>
<%
  Integer count = null;
  //同步处理
  synchronized (application)
  {
    //从内存当中读取访问量
    count = (Integer) application.getAttribute("basic.counter");
    if (count == null)
    count = new Integer(0);
    count = new Integer(count.intValue() + 1);
    //将访问量保存到内存当中
    application.setAttribute("basic.counter", count);
  }
%>
<html>
<head>
<title>简单计数器</title>
</head>
<body>
<center>
```

```
<font size=10 color=blue>简单计数器</font>
<br>
<hr>
<br>
<font size=5 color=blue>您好！您是本站的第 <%= count %> 位客人</font>
</center>
</body>
</html>
```

【运行结果】

运行结果如图 9-3 所示。



图 9-3

实例 97 汉字处理

本实例演示如何解决 JSP 初学者经常会碰到的汉字乱码问题。

【编程要点】

目前，大多数国际性的软件内部均采用 UNICODE 编码，在软件运行时，它获得本地支持系统（多数时间是操作系统）默认支持的编码格式，然后再将软件内部的 UNICODE 转化为本地系统默认支持的格式显示出来。Java 的 JDK 和 JVM 即是如此。由于 Java 语言内部采用 UNICODE 编码，所以在 Java 程序运行时，就存在着一个从 UNICODE 编码和对应的操作系统及浏览器支持的编码格式转换输入、输出的问题，这个转换过程有着一系列的步骤，如果其中任何一步出错，则显示出来的汉字就会出乱码，这就是常见的 Java 中文问题。

几乎所有的 Web 容器在其内部默认的字符编码格式都是以 ISO-8859-1 为默认值的，同时，几乎所有的浏览器在传递参数时都是默认以 UTF-8 的方式来传递参数的。所以，虽然 Java 源文件在出入口的地方指定了正确的编码方式，但其在容器内部运行时还是以 ISO-8859-1 来处理的。

【程序实现】

Input.html 文件内容如下：

```
<HTML>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=gb2312">
<HEAD>
<TITLE>汉字处理</TITLE>
</HEAD>
<BODY>
<CENTER>
<FONT SIZE = 5 COLOR = blue>汉字处理</FONT>
</CENTER>
<BR>
<HR>
```

```
<BR>
<!-- 以 Post 方法，将资料传递给 GB2312.jsp 档 -->
<FORM action="GB2312.jsp" method=post name=form>
    <BR>我是
    <INPUT type=text name=inputName>
    <BR>我的留言：
    <BR>
    <TEXTAREA rows=5 cols=30 name=txtaMsg></TEXTAREA>
    <INPUT type="submit" value="送出" name=submit>
    <INPUT type="reset" value="清除" name=reset>
</FORM>
</BODY>
</HTML>
```

GB2312.jsp 的文件内容如下：

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page language="java" %>
<HTML>
<HEAD>
<TITLE>汉字处理</TITLE>
</HEAD>
<BODY>
<CENTER>
<FONT SIZE = 5 COLOR = blue>汉字处理</FONT>
</CENTER>
<BR>
<HR>
<BR>
<H3>
<FONT COLOR = RED>
<CENTER>
<!--取得 Name 文本区的输入值-->
<%
    String tmpName = test.ISOtoGb2312.convert( request.getParameter("inputName") );
    out.println(tmpName);
%>
</FONT>
对大家说：
</CENTER>
<BR>
<FONT COLOR=BLUE>
<!--取得文字区的输入值-->
<%
    String tmpMsg = test.ISOtoGb2312.convert( request.getParameter("txtaMsg") );
    out.println(tmpMsg);
%>
</FONT>
</H3>
</BODY>
</HTML>
```

ISOtoGb2312.java 文件内容如下：

```
package test;
public class ISOtoGb2312
{
  public static String convert( String str )
  {
    try
    {
      byte[] bytesStr=str.getBytes( "ISO-8859-1" );
      return new String( bytesStr, "GB2312" );
    }
    catch( Exception ex)
    {
     return str;
    }
  }
}
```

【运行结果】

运行结果如图 9-4 所示。

单击“送出”后，结果如图 9-5 所示。



图 9-4



图 9-5

【分析与总结】

任何 Java 程序的生命期中，其编码转换的关键过程是在于：最初编译成 class 文件的编码和最终向用户输出的编码过程。

实例 98 文件的上传

本实例演示文件的上传的 JavaBean 实现。

【编程要点】

JavaBean 是描述 Java 的软件组件模型，有点类似于 Microsoft 的 COM 组件概念。在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，通过 JavaBean 的组合可以快速地生成新的应用程序。对于程序员来说，最好的一点就是 JavaBean 可以实现代码的重复利用，另外对于程序的易维护性等等也有很重大的意义。非可视化的 JavaBean，顾名思义就是没有 GUI 界面的 JavaBean。在 JSP 程序中常用来封装事务逻辑、数据库操作等等，可以很好地实现业务逻辑和前台程序（如 JSP 文件）的分离，使得系统具有更好的健壮性和灵活性。

通过 upload.jsp 调用 FileUpload 这个 Bean 中的 doUpload(request)方法上传选中的文件到指定的服务器目录下。

【程序实现】

upload.htm 文件的内容如下：

```
<html>
<head>
<title>文件上载</title>
</head>
<body>
<form action="upload.jsp" enctype="MULTIPART/FORM-DATA" method=post>
  <div align="center">上传者:
    <input type="text" name="name" />
    <br >
    上传说明:
    <input type="text" name="comment" >
    <br >
```

```
    选择要上载的文件
    <input type="file" name="filename" >
    <br >
    <input type="submit" value="上载" >
  </div>
</form>
</body>
</html>
```

upload.jsp 的文件内容如下：

```
<%@ page contentType="text/html;charSet=gb2312"%>
<HTML>
<HEAD>
<TITLE>
</TITLE>
</HEAD>
<BODY>
<%@ page language="java" import="test.FileUpload" %>
<jsp:useBean id="TheBean" scope="page" class="test.FileUpload"/>
<%
TheBean.doUpload(request);
out.println("<center>文件上传成功! ");
out.println("上传文件:" + TheBean.getFilename());
out.println("<BR>文件内容类型:" + TheBean.getContentType());
out.println("<BR>上传者:" + TheBean.getFieldValue("name"));
out.println("<BR>上传说明:" + TheBean.getFieldValue("comment"));
out.println("<center>");
%>
</BODY>
</HTML>
```

FileUpload.java 文件内容如下：

```
package test;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.ServletInputStream;
import java.util.Dictionary;
import java.util.Hashtable;
import java.io.*;

public class FileUpload {
    //定义变量，用来存储上传文件的名字、路径等
    private String savePath, filepath, filename, contentType;
    //保存了用户在表单中输入数据的名字/值对
    private Dictionary fields;
    //获得文件名字
    public String getFilename() {
        return filename;
    }
    //获得上传文件的路径
    public String getFilepath() {
        return filepath;
    }
    //设定上传文件的路径，如果没有设定就保存在服务器默认目录
    public void setSavePath(String savePath) {
        this.savePath = savePath;
    }
    //得到文件类型
    public String getContentType() {
        return contentType;
    }
    public String getFieldValue(String fieldName) {
        if (fields == null || fieldName == null)
            return null;
        return (String) fields.get(fieldName);
    }
    private void setFilename(String s) {
        if (s == null)
```

```
            return;
        int pos = s.indexOf("filename=\"");
        if (pos != -1) {
            filepath = s.substring(pos + 10, s.length() - 1);
            pos = filepath.lastIndexOf("\\");
            if (pos != -1)
                filename = filepath.substring(pos + 1);
            else
                filename = filepath;
        }
    }
    private void setContentType(String s) {
        if (s == null)
            return;
        int pos = s.indexOf(": ");
        if (pos != -1)
            contentType = s.substring(pos + 2, s.length());
    }
    public void doUpload(HttpServletRequest request) throws IOException {
        ServletInputStream in = request.getInputStream();
        byte[] line = new byte[128];
        int i = in.readLine(line, 0, 128);
        if (i < 3)
            return;
        int boundaryLength = i - 2;
        String boundary = new String(line, 0, boundaryLength); //-2 丢弃换行字符
        fields = new Hashtable();
        while (i != -1) {
            String newLine = new String(line, 0, i);
            if (newLine
                .startsWith("Content-Disposition: form-data; name=\"")) {
                if (newLine.indexOf("filename=\"") != -1) {
                    setFilename(new String(line, 0, i - 2));
                    if (filename == null)
                        return;
                    //文件内容
                    i = in.readLine(line, 0, 128);
                    setContentType(new String(line, 0, i - 2));
                    i = in.readLine(line, 0, 128);
                    //空行
                    i = in.readLine(line, 0, 128);
                    newLine = new String(line, 0, i);
                    PrintWriter pw =
                        new PrintWriter(
                            new BufferedWriter(
                                new FileWriter(
                                    (savePath == null ? "" : savePath)
                                        + filename)));
                    while (i != -1 && !newLine.startsWith(boundary)) {
                        //文件内容的最后一行包含换行字符
                        //因此我们必须检查当前行是否是最后一行
                        i = in.readLine(line, 0, 128);
                        if ((i == boundaryLength + 2
                            || i == boundaryLength + 4)
                            && (new String(line, 0, i).startsWith(boundary)))
                            pw.print(
                                newLine.substring(0, newLine.length() - 2));
                        else
                            pw.print(newLine);
                        newLine = new String(line, 0, i);
                    }
                    pw.close();
                } else {
                    //普通表单输入元素
                    //获取输入元素名字
                    int pos = newLine.indexOf("name=\"");
                    String fieldName =
                        newLine.substring(pos + 6, newLine.length() - 3);
                    i = in.readLine(line, 0, 128);
```

```
                    i = in.readLine(line, 0, 128);
                    newLine = new String(line, 0, i);
                    StringBuffer fieldValue = new StringBuffer(128);
                    while (i != -1 && !newLine.startsWith(boundary)) {
                        //最后一行包含换行字符
                        //因此我们必须检查当前行是否是最后一行
                        i = in.readLine(line, 0, 128);
                        if ((i == boundaryLength + 2
                            || i == boundaryLength + 4)
                            && (new String(line, 0, i).startsWith(boundary)))
                            fieldValue.append(
                                newLine.substring(0, newLine.length() - 2));
                        else
                            fieldValue.append(newLine);
                        newLine = new String(line, 0, i);
                    }
                    fields.put(fieldName, fieldValue.toString());
                }
            }
            i = in.readLine(line, 0, 128);
        }
    }
}
```

【运行结果】

在工程文件目录中创建一个 WEB-INF 目录，在其下创建一个 classes 文件，用于保存 Bean 的 class 文件，在 tomcat 的 conf 目录下修改 server.xml：

```
.........
<!-- Tomcat Manager Context -->
<Context path="/manager" docBase="manager" debug="0" privileged="true"/>
<!--下面是自己加入的-->
<Context path="/工程文件目录名" docBase="你的工程文件目录" debug="0" reloadable="true"/>
.........
```

运行结果如图 9-6 所示。

选择文件上传后，如图 9-7 所示。



图 9-6



图 9-7

【分析与总结】

通过 JavaBean 可以很好地实现逻辑的封装、程序的易于维护等等。

实例 99　获取客户机浏览器的信息

本实例演示如何获取客户机端的浏览器信息。

【编程要点】

JSP 中调用 Java 用户自定义类除了可以使用 JavaBean 外，另一种方式是在设置 JSP 页面的 import 属性。

【程序实现】

getBrowser.jsp 文件如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page language="java" %>
<%@ page import="test.**%>
<HTML>
<HEAD>
<TITLE>Get Browser</TITLE>
</HEAD>
<BODY>
<center>
<font size=5 color=blue>获得客户端浏览器信息</font>
<center>
<%
    Browser eins = new Browser(request, session);
%>
<br>
<hr>
<br>
<font size=4 face="隶书" color=blue>
您使用的浏览器是：<%=eins.getName()%>
</font>
<br>
<font size=r face="隶书" color=blue>
它是<%=eins.getCompany()%>公司的<%=eins.getVersion()%>版
</font>
<br>
<font size=4 face="隶书" color=blue>
您使用的操作系统是：<%=eins.getOs()%>
</font>
<br>
</BODY>
</HTML>
```

Browser.java 文件：

```
package test;

import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Browser extends HttpServlet
{
    protected HttpServletRequest request;
    protected HttpSession session;
    protected String userAgent;
    protected String company;        //公司名称
    protected String name;           //浏览器名称
    protected String version;        //版本信息
    protected String os;             //所使用的操作系统
    private Hashtable supportedLanguages; //Untersttzte Sprachen
    public Browser(HttpServletRequest request, HttpSession session)
    {
        this.initialize();
        this.request = request;
        this.session = session;
        this.setUserAgent(this.request.getHeader("User-Agent"));
        this.setCompany();
        this.setName();
        this.setVersion();
        this.setOs();
    }
    public void initialize()
    {
        this.supportedLanguages = new Hashtable(2);
        this.supportedLanguages.put("en", "");
        this.supportedLanguages.put("de", "");
    }
```

```
public void setUserAgent(String httpUserAgent)
{
    this.userAgent = httpUserAgent.toLowerCase();
}
//获取浏览器制造商的名称
private void setCompany()
{
    if (this.userAgent.indexOf("msie") > -1)
    {
        this.company = "Microsoft";
    }
    else if (this.userAgent.indexOf("opera") > -1)
    {
        this.company = "Opera Software";
    }
    else if (this.userAgent.indexOf("mozilla") > -1)
    {
    this.company = "Netscape Communications";
    }
    else
    {
    this.company = "unknown";
    }
}
public String getCompany()
{
    return this.company;
}
//获取浏览器的名称
private void setName()
{
    if (this.company == "Microsoft")
    {
        this.name = "Microsoft Internet Explorer";
    }
    else if (this.company == "Netscape Communications")
    {
        this.name = "Netscape Navigator";
    }
    else if (this.company == "Operasoftware")
    {
        this.name = "Operasoftware Opera";
    }
    else
    {
        this.name = "unknown";
    }
}
public String getName()
{
    return this.name;
}
//获取浏览器的版本
private void setVersion()
{
    int tmpPos;
    String tmpString;
    if (this.company == "Microsoft")
    {
        String str = this.userAgent.substring(this.userAgent.indexOf("msie") + 5);
        this.version = str.substring(0, str.indexOf(";"));
    }
    else
    {
     tmpString = (this.userAgent.substring(tmpPos = (this.userAgent.indexOf("/"))
                + 1, tmpPos + this.userAgent.indexOf(" "))).trim();
        this.version = tmpString.substring(0, tmpString.indexOf(" "));
    }
}
public String getVersion()
```

```
    {
        return this.version;
    }
    //获取操作系统制造商的名称
    private void setOs()
    {
        if (this.userAgent.indexOf("win") > -1)
        {
            if (this.userAgent.indexOf("windows 95") > -1
                    || this.userAgent.indexOf("win95") > -1)
            {
                this.os = "Windows 95";
            }
            if (this.userAgent.indexOf("windows 98") > -1
                    || this.userAgent.indexOf("win98") > -1)
            {
                this.os = "Windows 98";
            }
            if (this.userAgent.indexOf("windows nt") > -1
                    || this.userAgent.indexOf("winnt") > -1)
            {
                this.os = "Windows NT";
            }
            if (this.userAgent.indexOf("win16") > -1
                    || this.userAgent.indexOf("windows 3.") > -1)
            {
                this.os = "Windows 3.x";
            }
        }
    }
    public String getOs()
    {
        return this.os;
    }
}
```

【运行结果】

运行结果如图 9-8 所示。



图 9-8

【分析与总结】

通过本实例读者可以体会 JSP 与 JavaBean 实现方式的异同。

实例 100 收发邮件

本实例演示如何在 JSP 中收发邮件。

【编程要点】

本实例采用 Sun 公司提供的 JavaMail API 和 JavaBeans Activation Framework，下载后将 mail.jar 和 activation.jar 添加到 CLASSPATH 中去，或者将它们拷贝到 Tomcat 5.0\common\lib 下。

【程序实现】

JavaMail.html 文件内容如下：

```
<html>
<head>
<title>发送邮件</title>
</head>
<body bgcolor=#add8e6>
<CENTER>
<FONT SIZE=5 COLOR=BLUE>发送邮件</FONT>
</CENTER>
<BR>
<HR>
<BR>
<center>
<form name="email" action="sendMail.jsp" method="post" >
<table cellspacing="2" cellpadding="2" border="1">
<tr>
<td>收件人:</td>
<td><input type="text" name="to" size="30" maxlength="30"></td>
</tr>
<tr>
<td>发件人:</td>
<td><input type="text" name="from" size="30" maxlength="30" ></td>
</tr>
<tr>
<td>抄　送:</td>
<td><input type="text" name="cc" size="30" maxlength="100"></td>
</tr>
<tr>
<td>暗　送:</td>
<td><input type="text" name="bcc" size="30" maxlength="100"></td>
</tr>
<tr>
<td>主　题:</td>
<td><input type="text" name="subject" size="30" maxlength="30"></td>
</tr>
<tr>
<td colspan="2" align=center>
<textarea cols="40" rows="10" name="body"  >...</textarea>
</td>
</tr>
<tr>
<td  align=center colspan="2">
<input type="submit" name="submit" value="发送">    
<input type="Reset">
</td>
</tr>
</table>
</form>
</center>
</body>
</html>
```

sendMail.jsp 文件内容如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ page errorPage="errorPage.jsp" %>
<%@ page import="java.util.*" %>
<%@ page import="javax.mail.*" %>
<%@ page import="javax.mail.internet.*" %>
<%@ page import="javax.activation.*" %>
<html>
<head>
<title>发送邮件</title>
</head>
<body bgcolor=#add8e6>
<center>
```

```
    <font size=5 color=blue>发送邮件</font>
    </center>
    <BR>
    <HR>
    <BR>
    <%
    String host = "202.96.44.20";
    String to = request.getParameter("to");
    String from = request.getParameter("from");
    String cc= request.getParameter("cc");
    String bcc= request.getParameter("bcc");
    byte[] subjectTemp= request.getParameter("subject").getBytes("ISO8859_1");
    String subject=new String(subjectTemp);
    byte[] messageTextTemp = request.getParameter("body").getBytes("ISO8859_1");
    String messageText=new String(messageTextTemp);
    boolean sessionDebug = false;
    //validate the email address form.
    if ((to.trim().indexOf("@")==-1)||(to.trim().length()<5)){
    %>
    <jsp:forward page="error.jsp"/>
    <%}
    if ((from.trim().indexOf("@")==-1)||(from.trim().length()<5)){
    %>
    <jsp:forward page="error.jsp"/>
    <%}
    if
((cc.trim().length()>0&&cc.trim().length()<5)||((cc.trim().length()!=0)&&(cc.trim().in
dexOf("@")==-1)))
    {
    %>
    <jsp:forward page="error1.jsp"/>
    <%
    }
    if
((bcc.trim().length()>0&&bcc.trim().length()<5)||((bcc.trim().length()!=0)&&(bcc.trim(
).indexOf("@")==-1)))
    {
    %>
    <jsp:forward page="error1.jsp"/>
    <%
    }
    Properties props = System.getProperties();
    props.put("mail.host", host);
    props.put("mail.transport.protocol", "smtp");
    Session mailSession = Session.getDefaultInstance(props, null);
    mailSession.setDebug(sessionDebug);
    try{
    Message msg = new MimeMessage(mailSession);
    msg.setFrom(new InternetAddress(from));
    InternetAddress[] addressTo = {new InternetAddress(to)};
    msg.setRecipients(Message.RecipientType.TO,addressTo);
    if (cc.trim().length()!=0){
    InternetAddress[] addressCc = {new InternetAddress(cc)};
    msg.setRecipients(Message.RecipientType.CC,addressCc);
    }
    if (bcc.trim().length()!=0){
    InternetAddress[] addressBcc = {new InternetAddress(bcc)};
    msg.setRecipients(Message.RecipientType.BCC,addressBcc);
    }
    msg.setSubject(subject);
    msg.setSentDate(new Date());
    msg.setText(messageText);
    Transport.send(msg);
    }
    catch(Exception ex){
    out.println("sending failed");
    }%>
    <jsp:forward page="sendOk.jsp"/>
    </table>
    </body>
    </html>
```

sendOK.jsp 文件内容如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title>Your message was sent to destination successfully</title>
<head>
<body bgcolor=#add8e6>
<center>
<font size=5 color=blue>发送邮件</font>
</center>
<br>
<hr>
<br>
<center>
<h4>恭喜!你的邮件已成功发送到你朋友的信箱中了.<h4>
<table cellspacing="2" cellpadding="2" border="1">
<tr>
<td>朋友的信箱:</td><td><%=request.getParameter("to")%></td>
</tr><tr>
<td>邮件主题: :</td><td>
  <%
byte[] subjectTemp= request.getParameter("subject").getBytes("ISO8859_1");
String subject=new String(subjectTemp);
 out.println(subject);
%>
</td>
</tr>
<tr><td colspan="2" align=center><a href="javaMail.html" style="color: rgb(ff,0,0)">
返回</a>
</td></tr>
</table>
</center>
</body>
</html>
```

errorPage.jsp 文件内容如下：

```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>错误信息</title>
</head>
<center>
<font size=5 color=blue>错误信息</font>
</center>
<br>
<hr>
<br>
</center>
<body bgcolor=#add8e6>
     <h2><%= exception.getMessage() %></h2>
</body>
</center>
</html>
```

ERROR.jsp 文件内容如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title>错误信息</title>
<head>
<body bgcolor=#add8e6>
<center>
<font size=5 color=blue>错误信息</font>
</center>
<BR>
<HR>
<BR>
<center>
```

```
<h4>请检查电子邮件地址是否正确,且收件人地址不能为空!!<h4>
<br>
<a href="javaMail.html" style="color: rgb(ff,0,0)">返回</a>
</center>
</body>
</html>
```

ERROR1.jsp 文件内容如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title>错误信息</title>
<head>
<body bgcolor=#add8e6>
<center>
<font size=5 color=blue>错误信息</font>
</center>
<BR>
<HR>
<BR>
<center>
<h4>请检查电子邮件地址是否正确!!<h4>
<br>
<a href="javaMail.html" style="color: rgb(ff,0,0)">返回</a>
</center>
</body>
</html>
```

【运行结果】

运行结果如图 9-9 所示。

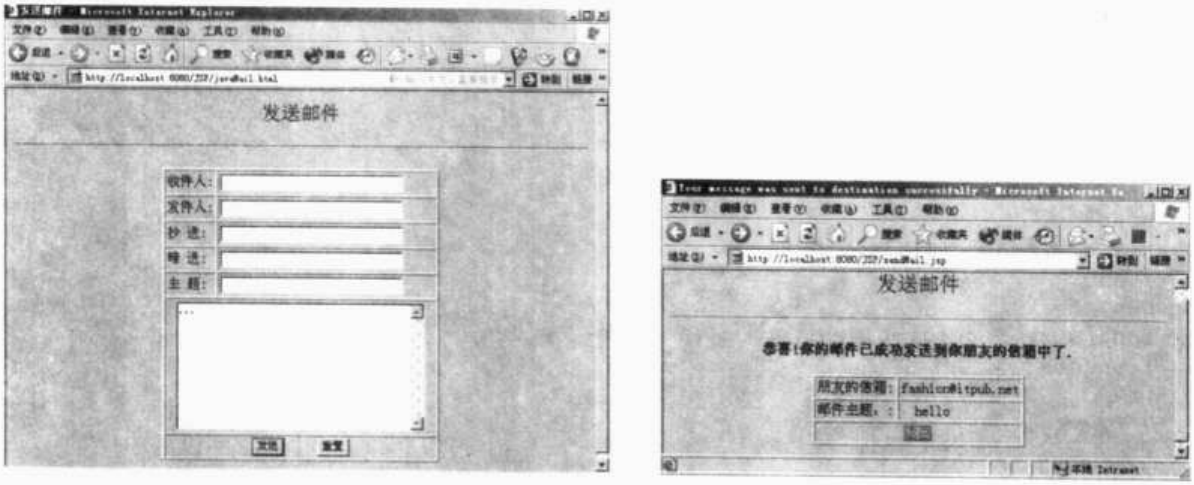


图 9-9

【分析与总结】

通过本实例可以了解到如何使用 JavaMail 的 API 来实现基本的收发 E-mail 功能。

实例 101 JSP 连接数据库

本实例演示如何在 JSP 中连接 SQLServer 2000 数据库。

【编程要点】

将下载下来的 JDBC driver 的 lib 目录下的 jar 文件拷贝到 tomcat 目录下的 common 的 lib 下。这里用到的数据库是以前用过的 book 数据库。

【程序实现】

testSQLServer.jsp 文件的内容如下：

```
<%@ page contentType="text/html;charset=gb2312"%>
<%@ page import="java.sql.*"%>
<html>
<body>
<%Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
String url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=book";
//book 为你的数据库的
String user="sa";  //访问数据库的用户名和密码
String password="sa";
Connection conn= DriverManager.getConnection(url,user,password);
Statement
stmt=conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
String sql="select * from Books";
ResultSet rs=stmt.executeQuery(sql);
while(rs.next()) {%>
您的第一个字段内容为：<%=rs.getString(1)%>
您的第二个字段内容为：<%=rs.getString(2)%>
<%}%>
<%out.print("数据库操作成功，恭喜你");%>
<%rs.close();
stmt.close();
conn.close();
%>
</body>
</html>
```

【运行结果】

运行结果如图 9-10 所示。



图 9-10

【分析与总结】

连接其他数据库如 Oracle、DB2、Informax 等数据库也类似。

实例 102 用 JavaBean 连接数据库

本实例演示如何利用 JavaBean 连接数据库。

【编程要点】

可以将数据库连接放在 JavaBean 中来实现，在 JSP 中只是通过调用 bean 就可以完成数据库的连接。

【程序实现】

DBBean.jsp 的文件内容如下：

```
<%@ page contentType="text/html; charset=GB2312" %>
<%@ page import="java.sql.*" %>
<%@ page language="java" %>
<HTML>
<HEAD>
<TITLE>用 Bean 链接数据库</TITLE>
</HEAD>
<BODY>
```

```
<CENTER>
<FONT SIZE = 5 COLOR = blue>
用 Bean 链接数据库
</FONT>
</CENTER>
<BR>
<HR>
<BR>
<CENTER>
<%--起始建立数据库链接的 Bean 对象--%>
<jsp:useBean id="ConBean" scope="session" class="beans.DBCon"/>
<%
//从 Bean 对象取得已完成建立的数据库链接
Connection con = ConBean.getConnection();
//建立 Statement 对象
Statement stmt = con.createStatement();
//建立 ResultSet(结果集)对象，并执行 SQL 叙述
ResultSet rs = stmt.executeQuery("SELECT ID, name FROM STUDENTS");
%>
<TABLE  bgcolor=DodgerBlue>
     <TR  bgcolor=SkyBlue>
     <TD><B>学 号</B></TD>
     <TD><B>姓 名</B></TD>
     </TR>
     <%
     //利用 while 循环将数据表中的记录列出
     while (rs.next())
     {
          %>
          <TR bgcolor=LightGoldenrodYellow>
          <TD><B><%= rs.getString("ID") %></B></TD>
          <TD><B><%= rs.getString("name") %></B></TD>
          </TR>
          <%
     }
     rs.close();              //关闭记录集
     stmt.close();            //关闭 Statement 对象
     ConBean.close();         //关闭数据库链接
%>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

DBCon.java 的文件内容如下：

```
package beans;

import java.io.*;
import javax.servlet.http.*;
import java.sql.*;

public class DBCon implements HttpSessionBindingListener
{
     private Connection con = null;       //与资料库连结有关的 Bean 属性
     public DBCon()                       //在建构子中完成资料库连结
     {
          BulidConnection();              //建立资料库连结
     }
     private void BulidConnection()       //建立资料库连结的方法
     {
          try{
            //载入驱动程式类别
               Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            //建立资料库连线
               con = DriverManager.getConnection("jdbc:odbc:student");
          }
```

```
        catch(Exception ex)
        {
            System.out.println(ex.toString());
        }
    }
    public Connection getConnection()
    {
        if(con == null)                    //若 con 为 null 时，重新建立资料库连结
            BulidConnection();
        return this.con;
    }
    public void close()
    {
        try{
            con.close();                   //关闭 Connection 物件
            con =  null;
        }
        catch(SQLException sex)
        {
            System.out.println(sex.toString());
        }
    }
      //当物件加入 session 时，将自动执行此函数
    public void valueBound(HttpSessionBindingEvent event){}
      //当 session 物件消灭时，将自动执行此函数
    public void valueUnbound(HttpSessionBindingEvent event)
    {
        if(con != null)
            close();                       //呼叫 close 方法
    }
}
```

【运行结果】

运行结果如图 9-11 所示。



图 9-11

【分析与总结】

通过此例可以进一步体会逻辑与页面分离的好处。

实例 103　网络定餐

本实例实现一个 B/S 结构的商业应用——网络定餐。

【编程要点】

JavaBeans 的属性与一般 Java 程序中所指的属性，或者说与所有面向对象的程序设计语言中。对象的属性是一个概念，在程序中的具体体现就是类中的变量。在 JavaBeans 设计中，按照属性的不同作用又细分为四类：Simple、Index、Bound 与 Constrained 属性。

（1）Simple 属性。Simple 属性表示伴随有一对 get/set 方法的变量。属性名与和该属

性相关的 get/set 方法名对应。例如：如果有 setX 和 getX 方法，则暗指有一个名为“X”的属性。如果有一个方法名为 isX，则通常暗指“X”是一个布尔属性（即 X 的值为 true 或 false）。

（2）Indexed 属性。Indexed 属性表示一个数组值。使用与该属性对应的 set/get 方法可取得数组中的数值。该属性也可一次设置或取得整个数组的值。

（3）Bound 属性。Bound 属性是指当该种属性的值发生变化时，要通知其他的对象。每次属性值改变时，这种属性就触发一个 PropertyChange 事件（在 Java 程序中，事件也是一个对象）。事件中封装了属性名、属性的原值、属性变化后的新值。这种事件传递到其他的 Beans，至于接收事件的 Beans 应做什么动作，由他自己定义。

（4）Constrained 属性。JavaBeans 的 Constrained 属性是指当这个属性的值要发生变化时，与这个属性已建立了某种连接的其他 Java 对象可否决属性值的改变。Constrained 属性的监听者通过抛出 Property VetoException 来阻止该属性值的改变。

假设 JavaBean 是在建立网站之前建好的，程序员要做的第一件事是告诉 JSP 网页它所需要使用 JavaBean.这工作可以用<jsp:useBean>卷标来完成：<jsp:useBean id="customer" class="beans.Customer" scope="request">。

<jsp:useBean>卷标需要程序员以 ID 属性来辨识 bean。在这里，程序员提供一个名称让 JSP 网页来辨识 bean，除了 ID 属性之外，也必须告诉网页要到哪里去找这个 bean，或者是它的 Java 类别名称。类别属性提供如何在各式方法之中找到它，最后一个需要的组件是 scope 属性有了范围属性的帮助，程序员可以告诉 bean，要它为单一网页(预设)[scope="page"]；为一个被请求的网页[scope="request"]；为通话期[scope="session"]；或为整个应用程序[scope="application"]来维护它自己的信息。对于通话期范围，你可以很容易的维护 JSP 网页里的项目，例如购物车。

对于下面这个 jsp 标签：<jsp:setProperty name="PepsiCola" property="name" param="PepsiColaName" />

Name 代表通过<jsp:useBean id="PepsiCola">标签定义的 JavaBean 对象实例。

Property 这是个很重要的属性，代表了程序员想设置值的属性 property 名字。如果使用 property="*"，程序就会反复的查找当前的 ServletRequest 所有参数，并且匹配 JavaBean 中相同名字的属性 property，并通过 JavaBean 中属性的 set 方法赋值 value 给这个属性。如果 value 属性为空，则不会修改 Javabean 中的属性值。

Param 属性代表了页面请求的参数名字，<jsp:setProperty>标签不能同时使用 param 和 value。

Value 属性代表了赋给 Bean 的属性 property 的具体值。

【程序实现】

webOrder.html 的文件内容如下：

```
<html>
<head>
<title>Web Order</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>

<body bgcolor=pink>
<form method="post" action="webOrder.jsp">
```

```
<center>
  <p><font size="5"><b><font size="6">网络订餐服务
  </font></b></font></p>
  <p><font size="3">客户名称:
  <input type="text" name="name" size="12" maxlength="12">
  电话:
  <input type="text" name="tel" size="12" maxlength="12">
  住址:
  <input type="text" name="addr" size="30" maxlength="30">
  </font></p>
</center>
<table width="90%" border="1" align="center" bgcolor="#CCCCCC">
  <tr>
    <td colspan="3" nowrap>
      <div align="center"><b>饮料</b></div>
    </td>
    <td colspan="2" nowrap>
      <div align="center"><b>薯条</b></div>
    </td>
    <td colspan="2" nowrap>
      <p align="center"><b>汉堡</b></p>
    </td>
    <td colspan="2" nowrap>
      <p align="center"><b>鸡块</b></p>
    </td>
    <td colspan="2" nowrap>
      <p align="center"><b>甜点</b></p>
    </td>
  </tr>
  <tr>
    <td nowrap>百事可乐</td>
    <td nowrap>
      <p>大(6)
        <input type="hidden" name="PepsiColaName" value="百事可乐大杯">
        <input type="hidden" name="PepsiColaPrice" value="6">
      </p>
      <p>小(5)
        <input type="hidden" name="PepsiColaName" value="百事可乐小杯">
        <input type="hidden" name="PepsiColaPrice" value="5">
      </p>
    </td>
    <td nowrap>
      <p>
        <input type="text" name="PepsiColaItem" size="3" maxlength="3">
        杯 </p>
      <p>
        <input type="text" name="PepsiColaItem" size="3" maxlength="3">
        杯 </p>
    </td>
    <td nowrap> 大(35)
      <input type="hidden" name="FrenchFriesName" value="薯条大份">
      <input type="hidden" name="FrenchFriesPrice" value="6">
      <p>小(25)
        <input type="hidden" name="FrenchFriesName" value="薯条小份">
        <input type="hidden" name="FrenchFriesPrice" value="4">
      </p>
    </td>
    <td nowrap>
      <p>
        <input type="text" name="FrenchFriesItem" size="3" maxlength="3">
        份</p>
      <p>
        <input type="text" name="FrenchFriesItem" size="3" maxlength="3">
        份</p>
```

```
    </td>
    <td nowrap width="18%">
      <p>麦辣鸡腿堡(10)
        <input type="hidden" name="SandwichesName" value="麦辣鸡腿堡">
        <input type="hidden" name="SandwichesPrice" value="10">
      </p>
    </td>
    <td nowrap width="6%">
      <input type="text" name="SandwichesItem" size="3" maxlength="3">
      个</td>
    <td nowrap width="15%">
      <p>吮指鸡块(4)
        <input type="hidden" name="NuggetsName" value="吮指鸡块">
        <input type="hidden" name="NuggetsPrice" value="4">
      </p>
    </td>
    <td nowrap width="9%">
      <input type="text" name="NuggetsItem" size="3" maxlength="3">
      个 </td>
    <td nowrap width="13%"> 冰淇淋(3)
      <input type="hidden" name="DessertName" value="冰淇淋">
      <input type="hidden" name="DessertPrice" value="3">
    </td>
    <td nowrap width="8%">
      <input type="text" name="DessertItem" size="3" maxlength="3">
      个 </td>
  </tr>
  <tr>
    <td nowrap> 七喜</td>
    <td nowrap>
      <p>大(6)
        <input type="hidden" name="SevenUPName" value="七喜大杯">
        <input type="hidden" name="SevenUPPrice2" value="5">
      </p>
      <p>小(5)
        <input type="hidden" name="SevenUPName" value="七喜小杯">
        <input type="hidden" name="SevenUPPrice" value="4">
      </p>
    </td>
    <td nowrap>
      <p>
        <input type="text" name="SevenUPItem" size="3" maxlength="3">
        杯 </p>
      <p>
        <input type="text" name="SevenUPItem" size="3" maxlength="3">
        杯 </p>
    </td>
    <td nowrap> </td>
    <td nowrap> </td>
    <td nowrap width="18%">
      <p>麦辣鱼块堡(10)
        <input type="hidden" name="SandwichesName2" value="麦辣鱼块堡">
        <input type="hidden" name="SandwichesPrice2" value="10">
      </p>
    </td>
    <td nowrap width="6%">
      <input type="text" name="SandwichesItem" size="3" maxlength="3">
      个 </td>
    <td nowrap width="15%">
      <p>上校鸡块(4)
        <input type="hidden" name="NuggetsName" value="上校鸡块">
        <input type="hidden" name="NuggetsPrice" value="4">
      </p>
    </td>
    <td nowrap width="9%">
```

```
    <input type="text" name="NuggetsItem" size="3" maxlength="3">
    个 </td>
  <td nowrap width="13%">
    <p>奶昔 (3)
      <input type="hidden" name="DessertName" value="奶昔">
      <input type="hidden" name="DessertPrice" value="3">
    </p>
  </td>
  <td nowrap width="8%">
    <input type="text" name="DessertItem" size="3" maxlength="3">
    份 </td>
</tr>
<tr>
  <td nowrap> 橙汁</td>
  <td nowrap>
    <p>大(4)
      <input type="hidden" name="OrangeJuiceName" value="橙汁大杯">
      <input type="hidden" name="OrangeJuicePrice" value="4">
    </p>
    <p>小(3)
      <input type="hidden" name="OrangeJuiceName" value="橙汁小杯">
      <input type="hidden" name="OrangeJuicePrice" value="3">
    </p>
  </td>
  <td nowrap>
    <p>
      <input type="text" name="OrangeJuiceItem" size="3" maxlength="3">
      杯 </p>
    <p>
      <input type="text" name="OrangeJuiceItem" size="3" maxlength="3">
      杯 </p>
  </td>
  <td nowrap> </td>
  <td nowrap> </td>
  <td nowrap width="18%">
    <p>麦辣鸡块堡(10)
      <input type="hidden" name="SandwichesName3" value="麦辣鸡快堡">
      <input type="hidden" name="SandwichesPrice22" value="10">
    </p>
  </td>
  <td nowrap width="6%">
    <input type="text" name="SandwichesItem" size="3" maxlength="3">
    个 </td>
  <td nowrap width="15%">
    <p>麦辣鸡翅(4)
      <input type="hidden" name="NuggetsName" value="麦辣鸡翅">
      <input type="hidden" name="NuggetsPrice" value="4">
    </p>
  </td>
  <td nowrap width="9%">
    <input type="text" name="NuggetsItem" size="3" maxlength="3">
    个 </td>
  <td nowrap width="13%">
    <p>苹果派(3)
      <input type="hidden" name="DessertName" value="苹果派">
      <input type="hidden" name="DessertPrice" value="3">
    </p>
  </td>
  <td nowrap width="8%">
    <input type="text" name="DessertItem" size="3" maxlength="3">
    份 </td>
</tr>
</table>
<p> </p><center>
  <input type="submit" name="Submit" value="确定">
```

```
    <input type="reset" name="Reset" value="重填">
  </center>
</form>
</body>
</html>
```

webOrder.jsp 的内容如下：

```
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page language="java" %>
<%@ page errorPage="orderErrorPage.jsp" %>
<%@ page import="java.util.*,beans.*;" %>
<jsp:useBean id="customer"
  class="beans.Customer" scope="request">
    <jsp:setProperty name="customer"
      property="*" />
</jsp:useBean>
<jsp:useBean id="PepsiCola"
    class="beans.Price" scope="request">
    <jsp:setProperty name="PepsiCola"
      property="name" param="PepsiColaName" />
    <jsp:setProperty name="PepsiCola"
      property="price" param="PepsiColaPrice" />
    <jsp:setProperty name="PepsiCola"
      property="item" param="PepsiColaItem" />
</jsp:useBean>
<jsp:useBean id="SevenUP"
    class="beans.Price" scope="request">
    <jsp:setProperty name="SevenUP"
      property="name" param="SevenUPName" />
    <jsp:setProperty name="SevenUP"
      property="price" param="SevenUPPrice" />
    <jsp:setProperty name="SevenUP"
      property="item" param="SevenUPItem" />
</jsp:useBean>
<jsp:useBean id="OrangeJuice"
    class="beans.Price" scope="request">
    <jsp:setProperty name="OrangeJuice"
      property="name" param="OrangeJuiceName" />
    <jsp:setProperty name="OrangeJuice"
      property="price" param="OrangeJuicePrice" />
    <jsp:setProperty name="OrangeJuice"
      property="item" param="OrangeJuiceItem" />
</jsp:useBean>
<jsp:useBean id="FrenchFries"
    class="beans.Price" scope="request">
    <jsp:setProperty name="FrenchFries"
      property="name" param="FrenchFriesName" />
    <jsp:setProperty name="FrenchFries"
      property="price" param="FrenchFriesPrice" />
    <jsp:setProperty name="FrenchFries"
      property="item" param="FrenchFriesItem" />
</jsp:useBean>
<jsp:useBean id="Sandwiches"
    class="beans.Price" scope="request">
    <jsp:setProperty name="Sandwiches"
      property="name" param="SandwichesName" />
    <jsp:setProperty name="Sandwiches"
      property="price" param="SandwichesPrice" />
    <jsp:setProperty name="Sandwiches"
      property="item" param="SandwichesItem" />
</jsp:useBean>
<jsp:useBean id="Nuggets"
    class="beans.Price" scope="request">
    <jsp:setProperty name="Nuggets"
      property="name" param="NuggetsName" />
    <jsp:setProperty name="Nuggets"
      property="price" param="NuggetsPrice" />
    <jsp:setProperty name="Nuggets"
      property="item" param="NuggetsItem" />
</jsp:useBean>
<jsp:useBean id="Dessert"
```

```
    class="beans.Price" scope="request">
    <jsp:setProperty name="Dessert"
      property="name" param="DessertName" />
    <jsp:setProperty name="Dessert"
      property="price" param="DessertPrice" />
    <jsp:setProperty name="Dessert"
      property="item" param="DessertItem" />
</jsp:useBean>
<html>
<head>
</head>
<body>
<center>
<%
    if( customer.getName().trim().equals("") ||
       customer.getTel().trim().equals("") ||
       customer.getAddr().trim().equals("") ){
        throw new ServletException( "请填写完整客户信息！" );
    }
%>
<p><font size="3">
    客户：<%=ISOtoGb2312.convert(customer.getName())%>
    电话：<%=customer.getTel()%>
    住址：<%=ISOtoGb2312.convert(customer.getAddr())%>
</font></p>
<hr width="50%">
<%
    int totalPrice = 0;
    Enumeration itemVecEmu = PepsiCola.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += PepsiCola.getTotalPrice();
    itemVecEmu = SevenUP.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += SevenUP.getTotalPrice();
    itemVecEmu = OrangeJuice.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += OrangeJuice.getTotalPrice();
    itemVecEmu = FrenchFries.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += FrenchFries.getTotalPrice();
    itemVecEmu = Sandwiches.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += Sandwiches.getTotalPrice();
    itemVecEmu = Nuggets.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= ISOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += Nuggets.getTotalPrice();
```

```
    itemVecEmu = Dessert.getItemList().elements();
    while( itemVecEmu.hasMoreElements() ){
%>
        <p><%= iSOtoGb2312.convert(itemVecEmu.nextElement().toString()) %></p>
<%
    }
    totalPrice += Dessert.getTotalPrice();
%>
<hr width="50%">
<%
    if( totalPrice==0 ){
        throw new ServletException( "没有订购任何快餐！" );
    }
%>
<%= "总价是: " + totalPrice %>
</center>
</body>
</html>
```

Customer.java 的文件内容如下：

```
package beans;
public class Customer{

    private String name="";
    private String tel="";
    private String addr="";

    public Customer(){
    }
    public void setName( String name ){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public void setTel( String tel ){
        this.tel = tel;
    }
    public String getTel(){
        return this.tel;
    }
    public void setAddr( String addr ){
        this.addr = addr;
    }
    public String getAddr(){
        return this.addr;
    }
}
```

Price.java 文件内容如下：

```
package beans;
import java.util.Vector;

public class Price{
    private String[] name;
    private int[] price;
    private String[] item;

    public Price(){
    }

    public void setName( String[] name ){
        this.name = name;
    }

    public String[] getName(){
        return this.name;
    }

    public void setPrice( int[] price ){
        this.price = price;
```

```
    }

    public int[] getPrice(){
        return this.price;
    }

    public void setItem( String[] item ){
        this.item = item;
    }

    public String[] getItem(){
        return this.item;
    }

    //获取商品的列表
    public Vector getItemList(){
        Vector itemVec = new Vector();
        for( int num=0; num<item.length; num++  )
          if( !item[num].trim().equals( "" ) )
            try{
             int items = Integer.parseInt(item[num].trim());
             if( items > 0 ){
                 itemVec.addElement( name[num] + " X " + item[num].trim() +
                 " = $" + (items*price[num]) );
              }
            }
            catch( NumberFormatException numex ){
            }

        return itemVec;
    }

    //获取商品的总价
    public int getTotalPrice(){
        int totalprice=0;
        for( int num=0; num<item.length; num++  )
          if( !item[num].trim().equals( "" ) )
            try{
             int items = Integer.parseInt(item[num].trim());
             if( items > 0 )
                 totalprice += items*price[num];
            }
            catch( NumberFormatException numex ){
            }
        return totalprice;
    }

}
```

ISOtoGb2312.java 的文件内容如下：

```
package beans;

public class ISOtoGb2312
{

  public static String convert( String str )
  {
     try
     {
      byte[] bytesStr=str.getBytes( "ISO-8859-1" );
      return new String( bytesStr, "GB2312" );
     }
     catch( Exception ex)
     {
     return str;
     }
  }
}
```

【运行结果】

运行结果如图 9-12 所示。



图 9-12

提交后，返回结果如图 9-13 所示。



图 9-13

【分析与总结】

JavaBean 的价值在于它可以经由一组特性来使用，而这些特性则提供对 JavaBean 设定的存取。以人为例，此人就是 JavaBean，而他的姓名、社会福利安全号码以及住址可以是属性。对于 JSP 网站，基本上是将 JavaBean 动态的连接到网站。

实例 104 简单的 Servlet 程序

本实例演示如何在 Java 的应用服务器 tomcat 上进行 servlet 编程。

【编程要点】

Servlet 是一种独立于平台和协议的服务器端的 Java 应用程序，可以生成动态的 Web 页面。Servlet 是位于 Web 服务器内部的服务器端的 Java 应用程序，与传统的从命令行启动的 Java 应用程序不同，Servlet 由 Web 服务器进行加载，该 Web 服务器必须包含支持 Servlet 的 Java 虚拟机。

Servlet 和 Applet 的区别，Servlet 是用于服务器上的、扩展服务器功能的 Java 程序；而 Applet 是在浏览器端执行的 Java 小程序。

用 Servlet 替代 CGI Script Servlet，提供了很简单地生成动态 HTML 页面的功能和比 CGI 执行得更快的运行效率。

一个 Servlet 能够同时处理多个请求，并能保持请求同步，这使 Servlet 能够支持在线会议方式。Servlet 通过转发请求到其他的 Servers 和 Servlets，能够在几个服务器间平衡负载。

HttpServlet 是从 GenericServlet 继承而来，是使用 Servlet 编程经常用到的包，它支持 HTTP 的 post 和 get 等方法。

【程序实现】

Test.html 页面文件的内容如下：

```
<html>
<head>
<title>简单 Servlet 程序 </title>
</head>
<body bgcolor="white">
<table border="2" cellpadding="2" align="CENTER">
<tr><td width="275">
<H2> 计算整数的阶乘</H2>
<br> 点击 "计算"，调用 Servlet <br>
<!调用 WEB-INF/classes/TestServlet.class>
<form method="post" action="TestServlet">
<input type="text" name="DATA" size=30>
<p>
<input type="SUBMIT" value= "计算">
<input type="RESET" value= "重置">
</form>
</td></tr>
</table>
</body>
</html>
```

服务器端 TestServlet.java 文件的内容如下：

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TestServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html; charset=GBK";
    private int number;
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter(); //服务器向客户端输出的输出流
        int n = 1;
        //从客户端得到对象 DATA 的值
        number = Integer.parseInt(request.getParameter("DATA"));
        for (int i = 1; i <= number; i++) {
            n *= i;
        }
        out.println("<h2 align='center'>" + "计算结果" + "</h2>");
        out.println("<html>");
        out.println("<head><title>TestServlet</title></head>");
        out.println("<body>");
        out.println("<P>"+number +"! = " + n + "</P>");
        out.println("</body></html>");
        out.close();
    }
}
```

【运行结果】

要使 Servlet 能被正常地调用，还需配置 web.xml 文件，创建 WEB-INF 文件夹，在该文件夹下创建 web.xml，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
```

```
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
    <web-app>
    <servlet>
    <servlet-name>TestServlet</servlet-name>
    <servlet-class>TestServlet</servlet-class>
    </servlet>
    <servlet-mapping>
    <servlet-name>TestServlet</servlet-name>
    <url-pattern>/TestServlet</url-pattern>
    </servlet-mapping>
    </web-app>
```

其中 url-pattern 标签指明了该 Servlet 的网络路径，HTML 调用它时要注意这个路径。然后将整个文件夹(该文件夹包含 text.html 和 WEB-INF 文件夹及其下的 web.xml 和 classes 文件夹，classes 文件夹包含 TextServlet.class 文件) 发布到 Tomcat 服务器上。

运行结果如图 9-14 所示。

输入想要计算的整数后，结果如图 9-15 所示。



图 9-14



图 9-15

【分析与总结】

这里用到了 doPost 方法，用 HttpServletRequest 对象的 getParameter()方法来从表单中获取数据。因为它返回数据给客户端，doPost()调用 getWriter()方法，然后用 out.println() 方法将输出 html 格式的数据。

实例 105　在 Servlet 中实时地创建图像

本实例演示如何在 Servlet 中创建 JPEG 图像。

【编程要点】

在 Java 创建图像或进行图像处理，要直接从 Servlet 输出图像时，通过给 image/gif(for GIFs) 或 image/jpeg (for JPEGs) 设置 ContentType 头来实现它。然后，打开 HttpResponse Output 流作为原始流，而不是打印流，用 write()方法直接传送字节。

【程序实现】

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import com.sun.image.codec.jpeg.*;
import java.awt.image.*;
import java.awt.*;

public class JPEGServlet extends HttpServlet{
    //处理 HTTP Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException,IOException{
```

```
            response.setContentType("image/jpeg");
            ServletOutputStream out= response.getOutputStream();
            BufferedImage      image      =      new      BufferedImage(100,      100,
BufferedImage.TYPE_INT_RGB);
            Graphics g = image.getGraphics();
            g.setColor(Color.green);
            g.fillRect(0,0,100,100);
            g.setColor(Color.red);
            g.drawOval(0,0,100,100);
            JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
            encoder.encode(image);
            out.close();
        }
        //处理 HTTP Post 请求
        public void doPost(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
           doGet(request,response);
        }
        //得到 Servlet 信息
        public String getServletInfo() {
            return "JPEGServlet Information";
        }
    }
```

【运行结果】

运行结果如图 9-16 所示。



图 9-16

在一些 Servlet 引擎设置中，Servlet 的目录不能通过 Web Server 进入，只能通过 Servlet 引擎，也就是说不能通过 http://URL 登录，可以向 Servlet 输出的 HTML 传送 IMG 标签，或传送 HTTP 重新定位来让浏览器直接下载图像。

【分析与总结】

图像可以保存在浏览器的 Cache 中，当再次请求时不必重新运行 Servlet，因此减轻了服务器的负担。

图像不能从磁盘中删除，因此必须写一段程序来定期清理图像目录，或进入目录后手工删除。

实例 106　简单购物车

本实例实现一个 Servlet 的应用——购物车。

【编程要点】

使用 Java Servlet API 的会话对象为每个用户提供了跟踪会话的方式，其中有一种方式的实现是利用 HttpSession 对象，HttpSession 对象用来在当前 servlet 环境中存储会话数据，HttpSession 对象由 HttpSession 接口定义，HttpSession 对象可以使用 HttpServletRequest 对象的 getSession()方法获得。

HttpSession 对象的方法列表：

long getCreationTime()：返回会话创建时间。

String getID()：返回会话 ID。

long getLastAccessedTime()：返回上一次携带这个 ID 的请求时间。

void invalidate()：删除当前会话。

boolean isNew()：如果会话已创建并未发送到浏览器返回 true。

String[] getValueNames()：返回为本次会话存储的所有值的名字数组 object。

getValue(string name)：返回给定名字的值。

void putValue(String name,object value)：将一个值增加到会话。

void removeValue(String name)：从会话中删除一个条目。

int setMaxInactiveInterval(int interval)：设置请求之间的会话存活时间。

int getMaxInactiveInterval()：返回请求之间的会话存活时间。

【程序实现】

shop.html 的文件内容如下：

```
<html>
<head>
<title>Shop</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>

<body bgcolor="#FFFFFF">
<h1 align="center">Book Shop</h1>
<hr>
<p><b>choose following information:</b></p>
<form method="post" action="ShoppingCart">
  <table width="500" border="0" cellspacing="0" cellpadding="0">
    <tr>
      <td width="69">
        <input type="checkbox" name="item" value="Thinking in Java">
      </td>
      <td width="431">Item1:Thinking in Java</td>
    </tr>
    <tr>
      <td width="69">
        <input type="checkbox" name="item" value="Java Programming Language SL-275">
      </td>
      <td width="431">Item2:Java Programming Language SL-275</td>
    </tr>
    <tr>
      <td width="69">
        <input type="checkbox" name="item" value="Beancontext">
      </td>
      <td width="431">Item3:Beancontext</td>
    </tr>
  </table>
  <hr>
  <p>
    <input type="submit" name="btn_submit" value="Buy Now">
  </p>
</form>
<p> </p>
</body>
</html>
```

ShoppingCart.java 的文件内容如下：

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
public class ShoppingCart extends HttpServlet {
        public void doPost(HttpServletRequest req,HttpServletResponse res) throws
ServletException ,IOException{
                //取得 Session 对象
                //如果 Session 不存在，为本次会话创建此对象
                HttpSession session =req.getSession(true);
                Integer itemCount=(Integer)session.getValue("itemCount");
                //如果 session 是新的
                if (itemCount==null)
                        itemCount=new Integer(0);
                PrintWriter out=res.getWriter();
                res.setContentType("text/html");
                //接收传来的参数
                String[] itemsSelected;
                String itemName;
                itemsSelected=req.getParameterValues("item");
                if(itemsSelected !=null){
                        for(int i=0;i<itemsSelected.length;i++){
                                itemName=itemsSelected[i];
                                System.out.println(itemName);
                                itemCount=new Integer(itemCount.intValue()+1);
                                //购买的条目
                                session.putValue("item"+itemCount,itemName);
                                //总条目
                                session.putValue("itemCount",itemCount);
                        }
                }
                out.println("<html>");
                out.println("<title>");
                out.println("item list");
                out.println("</title>");
                out.println("<body><h4>Session List:</h4><hr><br><br>");
                for(int i=1;i<=itemCount.intValue();i++){
                        out.println((String)session.getValue("item"+i)+"<hr>");
                }
                out.println("</body>");
                out.println("</html>");
                out.close();
        }
}
```

【运行结果】

运行结果如图 9-17 所示。

当顾客单击“Buy Now”按钮后，将请求提交给一个 Servlet，Servlet 程序用来对顾客的操作进行处理，记录下顾客所购买的商品，并显示出目前顾客将要购买的所有商品清单，如图 9-18 所示。

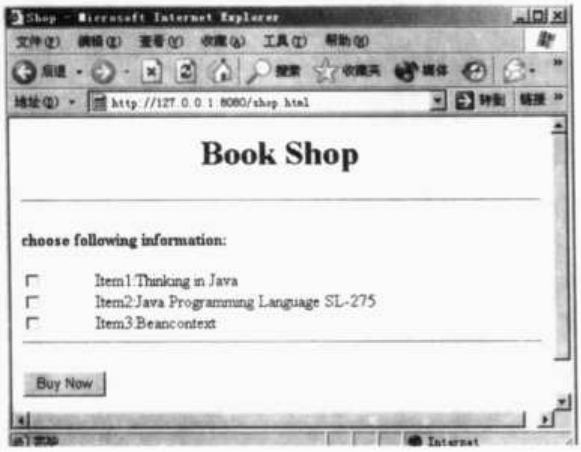


图 9-17

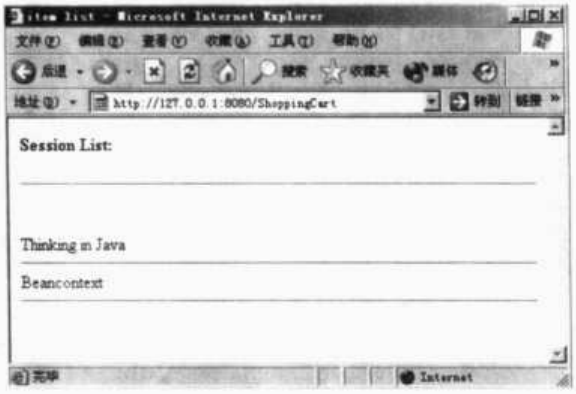


图 9-18

【分析与总结】

本例实现了一个简单的购物车程序，目的是通过购物车程序让读者了解 HttpSession 类在 servlet 中的用法和实现一个购物车的基本思路。

实例 107 用户注册登录系统

本实例用 JSP+JavaBean+Servlet 技术编写一个网站常用的用户注册登录系统。

【编程要点】

结合上面的例子，分别完成数据库 Bean，事务逻辑的 Bean（即用户注册、验证 Bean）和注册请求的 Servlet。这里用到 SQLServer 的 Reg 数据库，含有一个表 Users，记录用户的注册信息：id(int)、name(vchar)、pwd(vchar)、email(vchar)。

Cookie 是服务器发送给浏览器的体积很小的纯文本信息，用户以后访问同一个 Web 服务器时浏览器会把它们原样发送给服务器。通过让服务器读取它原先保存到客户端的信息，网站能够为浏览者提供一系列的方便，例如在线交易过程中标识用户身份、安全要求不高的场合避免用户重复输入名字和密码、门户网站的主页定制、有针对性地投放广告等等。

把 Cookie 发送到客户端时，Servlet 先要调用 new Cookie(name,value)用合适的名字和值创建一个或多个 Cookie，通过 cookie.setXXX 设置各种属性，通过 response.addCookie(cookie)把 cookie 加入应答头。

要从客户端读入 Cookie，Servlet 应该调用 request.getCookies()，getCookies()方法返回一个 Cookie 对象的数组。在大多数情况下，只需要用循环访问该数组的各个元素寻找指定名字的 Cookie，然后对该 Cookie 调用 getValue 方法取得与指定名字关联的值。

【程序实现】

连接数据库的 JavaBean。DBconn.java 文件内容如下：

```
//DBConn.java
package beans;
import java.sql.*;

public class DBConn {
    public String sql_driver = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    public String sql_url = "jdbc:microsoft:sqlserver://192.168.29.226:1433";
    public String sql_DBName = "Reg";
    public String user = "sa";
    public String pwd = "sa";
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    public boolean setDriver(String drv) {
        this.sql_driver = drv;
        return true;
    }
    public String getDriver() {
        return this.sql_driver;
    }
    public boolean setUrl(String url) {
        this.sql_url = url;
        return true;
    }
    public boolean setDBName(String dbname) {
        this.sql_DBName = dbname;
        return true;
    }
    public String getDBName() {
```

```
        return this.sql_DBName;
    }
    public boolean setUser(String user) {
        this.user = user;
        return true;
    }
    public String getUser() {
        return this.user;
    }
    public boolean setPwd(String pwd) {
        this.pwd = pwd;
        return true;
    }
    public String getPwd() {
        return this.pwd;
    }
    public DBConn() {
        try {
            Class.forName(sql_driver); //加载数据库驱动程序
            this.conn =
                DriverManager.getConnection(
                    sql_url + ";DatabaseName=" + sql_DBName,user,pwd);
            this.stmt = this.conn.createStatement();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
    //执行查询操作
    public ResultSet executeQuery(String strSql) {
        try {
            this.rs = stmt.executeQuery(strSql);
            return this.rs;
        } catch (SQLException e) {
            System.out.println(e.toString());
            return null;
        } catch (NullPointerException e) {
            System.out.println(e.toString());
            return null;
        }
    }
    //执行数据的插入、删除、修改操作
    public boolean execute(String strSql) {
        try {
            if (this.stmt.executeUpdate(strSql) == 0)
                return false;
            else
                return true;
        } catch (SQLException e) {
            System.out.println(e.toString());
            return false;
        } catch (NullPointerException e) {
            System.out.println(e.toString());
            return false;
        }
    }
    //结果集指针跳转到某一行
    public boolean rs_absolute(int row) {
        try {
            this.rs.absolute(row);
            return true;
        } catch (SQLException e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public void rs_afterLast() {
        try {
            this.rs.afterLast();
        } catch (SQLException e) {
            System.out.println(e.toString());
```

```
        }
    }
    public void rs_beforeFirst() {
        try {
            this.rs.beforeFirst();
        } catch (SQLException e) {
            System.out.print(e.toString());
        }
    }
    public void rs_close() {
        try {
            this.rs.close();
        } catch (SQLException e) {
            System.out.print(e.toString());
        }
    }
    public void rs_deleteRow() {
        try {
            this.rs.deleteRow();
        } catch (SQLException e) {
            System.out.print(e.toString());
        }
    }
    public boolean rs_first() {
        try {
            this.rs.first();
            return true;
        } catch (SQLException e) {
            System.out.print(e.toString());
            return false;
        }
    }
    public String rs_getString(String column) {
        try {
            return this.rs.getString(column);
        } catch (SQLException e) {
            System.out.println(e.toString());
            return null;
        }
    }
    //此方法用于获取大段文本，将其中的回车换行替换为<br>
    //输出到 html 页面
    public String rs_getHtmlString(String column) {
        try {
            String str1 = this.rs.getString(column);
            String str2 = "\r\n";
            String str3 = "<br>";
            return this.replaceAll(str1, str2, str3);
        } catch (SQLException e) {
            System.out.println(e.toString());
            return null;
        }
    }
    //把 str1 字符串中的 str2 字符串替换为 str3 字符串
    private static String replaceAll(String str1, String str2, String str3) {
        StringBuffer strBuf = new StringBuffer(str1);
        int index = 0;
        while (str1.indexOf(str2, index) != -1) {
            index = str1.indexOf(str2, index);
            strBuf.replace(
                str1.indexOf(str2, index),
                str1.indexOf(str2, index) + str2.length(),
                str3);
            index = index + str3.length();
            str1 = strBuf.toString();
        }
        return strBuf.toString();
    }
    public int rs_getInt(String column) {
        try {
```

```
            return this.rs.getInt(column);
        } catch (SQLException e) {
            System.out.println(e.toString());
            return -1;
        }
    }
    public int rs_getInt(int column) {
        try {
            return this.rs.getInt(column);
        } catch (SQLException e) {
            System.out.println(e.toString());
            return -1;
        }
    }
    public boolean rs_next() {
        try {
            return this.rs.next();
        } catch (SQLException e) {
            System.out.println(e.toString());
            return false;
        }
    }
    //判断结果集中是否有数据
    public boolean hasData() {
        try {
            boolean has_Data = this.rs.first();
            this.rs.beforeFirst();
            return has_Data;
        } catch (SQLException e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean rs_last() {
        try {
            return this.rs.last();
        } catch (SQLException e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean rs_previous() {
        try {
            return this.rs.previous();
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
}
```

用户注册 Bean。reg.java 文件内容如下：

```
//reg.java
package beans;
import java.sql.*;

public class reg {
    public int newID = 0;
    public boolean result = false;
    public boolean reg(
        String username,
        String password,
        String confirm,
        String email) {
        try {
            if (!this.checkUser(username))
                return false;
            if (!this.checkPwd(password))
                return false;
            if (!this.verifyPwd(password, confirm))
                return false;
```

```
            if (!this.checkEmail(email))
                return false;
            if (!this.userNotExit(username))
                return false;
            this.getNewID();
            this.result = this.register(username, password, confirm, email);
            return this.result;
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean checkUser(String user) {
        try {
            if (user.indexOf("'") != -1) {
                System.out.println("姓名中含有非法字符! ");
                return false;
            } else
                return true;
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean checkPwd(String pwd) {
        try {
            if (pwd.indexOf("'") != -1) {
                System.out.println("密码中含有非法字符! ");
                return false;
            } else
                return true;
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean verifyPwd(String pwd, String confirm) {
        try {
            if (!pwd.equals(confirm)) {
                System.out.println("两次输入的密码不一致! ");
                return false;
            } else
                return true;
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean checkEmail(String email) {
        try {
            if (email.indexOf("'") != -1) {
                System.out.println("E-mail 中含有非法字符! ");
                return false;
            } else
                return true;
        } catch (Exception e) {
            System.out.println(e.toString());
            return false;
        }
    }
    public boolean userNotExit(String user) {
        try {
            DBConn userDBConn = new DBConn();
            userDBConn.executeQuery(
                "select * from Users where name='" + user + "'");
            if (userDBConn.rs_next()) {
                System.out.println("用户名已存在，请选择其他的用户名! ");
                return false;
            } else
```

```
                return true;
            } catch (Exception e) {
                System.out.println(e.toString());
                return false;
            }
        }
        public int getNewID() {
            try {
                DBConn newIDDBConn = new DBConn();
                newIDDBConn.executeQuery("select * from Users order by id desc");
                if (newIDDBConn.rs_next()) {
                    this.newID = newIDDBConn.rs_getInt("id") + 1;
                    System.out.println(this.newID);
                } else {
                    this.newID = 1;
                }
                return this.newID;
            } catch (Exception e) {
                System.out.println(e.toString());
                return -1;
            }
        }
        public int getID() {
            return this.newID;
        }
        public boolean register(
            String username,
            String password,
            String confirm,
            String email) {
            try {
                DBConn regDBConn = new DBConn();
                String strSQL =
                    "insert into Users(id,name,pwd,email) values('"
                        + this.newID
                        + "','"
                        + username
                        + "','"
                        + password
                        + "','"
                        + email
                        + "')";
                regDBConn.execute(strSQL);
                return true;
            } catch (Exception e) {
                System.out.println(e.toString());
                return false;
            }
        }
    }
```

用户登录 Servlet。login.java 文件内容如下：

```
//login.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import beans.DBConn;
import java.sql.*;

public class login extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        if (this.checklogin(username, password)) {
            Cookie mylogin = new Cookie("username", username);
            mylogin.setVersion(1);
            mylogin.setPath("/");
            mylogin.setComment("Your login username");
            res.addCookie(mylogin);
        }
```

```
            //修改重定向的URL，指向首页
            res.sendRedirect("/Login/index.jsp");
        }
        public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws IOException, ServletException {
            doGet(req, res);
        }
        public static String getCookieValue(
            Cookie[] cookies,
            String cookieName,
            String defaultValue) {
            for (int i = 0; i < cookies.length; i++) {
                Cookie cookie = cookies[i];
                if (cookieName.equals(cookie.getName()))
                    return (cookie.getValue());
            }
            return (defaultValue);
        }
        public boolean checklogin(String username, String password) {
            try {
                DBConn loginConn = new DBConn();
                loginConn.executeQuery(
                    "select * from Users where name='" + username + "'");
                if (loginConn.rs_next()) {
                    System.out.println("Connection created!");
                    if (loginConn.rs_getString("pwd").trim().equals(password)) {
                        System.out.println(loginConn.rs_getString("name"));
                        return true;
                    } else {
                        return false;
                    }
                }
                return false;
            }
            catch (Exception e) {
                System.out.println(e.toString());
                return false;
            }
        }
}
```

检测用户是否已经登录的 Bean。checkLogin.java 文件内容如下：

```
//checkLogin.java
package beans;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class checkLogin {
    public String username = "";
    public boolean check(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {
        String cookieName = "username";
        PrintWriter out = res.getWriter();
        try {
            Cookie[] myCookies = req.getCookies();
            this.username =
                this.getCookieValue(myCookies, cookieName, "not found");
        }
        catch (Exception e) {
            return false;
        }
        if (!this.username.equals(new String("not found"))) {
            return true;
        }
        else {
            return false;
        }
    }
    public String getUserName() {
        return this.username;
```

```
        }
        public static String getCookieValue(
             Cookie[] cookies,
             String cookieName,
             String defaultValue) {
             for (int i = 0; i < cookies.length; i++) {
                  Cookie cookie = cookies[i];
                  if (cookieName.equals(cookie.getName()))
                       return (cookie.getValue());
             }
             return (defaultValue);
        }
    }
```

主页 index.jsp 文件内容如下：

```
<%@ page contentType="text/html;charset=gb2312" pageEncoding="gb2312" %>
<html>
<head>
<title>Java 世界</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0">
<jsp:useBean id="checklogin" class="beans.checkLogin" scope="page"/>
<%
   boolean login = checklogin.check(request,response);
%>
<table width="100%" height="100%" border="0" cellpadding="0" cellspacing="0">
  <tr bgcolor="green">
    <td height="80" colspan="5"><table width="100%" height="100%" border="0"
cellpadding="0" cellspacing="0">
        <tr>
          <td width="120"> </td>
          <td class="caption">Java 世界</td>
          <td width="200"> </td>
        </tr>
      </table></td>
  </tr>
  <tr>
    <td width="200" align="center" valign="top"><table width="100%" height="20"
border="0" cellpadding="0" cellspacing="0">
        <tr>
          <td> </td>
        </tr>
      </table>
    <%
     if(!login){
    %>
      <table width="90%" border="0" align="center" cellpadding="0" cellspacing="1"
bgcolor="#CCCCCC">
    <form name="form1" method="post" action="login">
        <tr align="center" bgcolor="#CCCCCC">
          <td height="30" colspan="2" class="deepred">用户登录</td>
        </tr>
        <tr>
          <td width="50%" height="24" align="center" bgcolor="#FFFFFF">用户名</td>
          <td align="center" bgcolor="#FFFFFF"><input name="username" type="text"
id="username" size="10"></td>
        </tr>
        <tr>
          <td height="24" align="center" bgcolor="#FFFFFF">密码</td>
          <td align="center" bgcolor="#FFFFFF"><input name="password" type="password"
id="password" size="10"></td>
        </tr>
        <tr>
          <td height="24" align="center" bgcolor="#FFFFFF"><a href="reg.jsp"
target="_blank" class="red">注册</a></td>
          <td align="center" bgcolor="#FFFFFF"><input type="submit" name="Submit"
value="进入"></td>
        </tr>
```

```
      </form>
         </table>
      <%
        }
      else
      {
       out.println("您好, " + checklogin.getUserName() + "!");
      }
       %>
       </td>
     <td width="1" valign="top" bgcolor="#CCCCCC"></td>
        <td width="400"> </td>
     <td width="1" valign="top" bgcolor="#CCCCCC"></td>
        <td width="200"> </td>
      </tr>
      <tr align="center" bgcolor="green">
        <td height="60" colspan="5" class="white">欢迎来到 Java 世界</td>
      </tr>
    </table>
    </body>
    </html>
```

注册页面 reg.jsp 文件内容如下：

```
    <%@ page contentType="text/html;charset=gb2312" pageEncoding="gb2312" %>
    <html>
    <head>
    <title>Java 世界</title>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    </head>
    <body bgcolor="#FFFFFF" leftmargin="0" topmargin="0">
    <table width="100%" height="100%" border="0" cellpadding="0" cellspacing="0">
      <tr bgcolor="green">
        <td  height="80"  colspan="5"><table  width="100%"  height="100%"  border="0"
cellpadding="0" cellspacing="0">
            <tr>
              <td width="120"> </td>
              <td class="caption">Java 世界</td>
              <td width="200"> </td>
            </tr>
          </table></td>
      </tr>
      <tr>
        <td width="100" align="center" valign="top"> </td>
        <td width="1" valign="top"></td>
        <td  width="400"  align="center"  valign="top"><table  width="100%"  height="20"
border="0" cellpadding="0" cellspacing="0">
            <tr>
              <td> </td>
            </tr>
          </table>
          <table     width="100%"     border="0"     cellpadding="0"     cellspacing="1"
bgcolor="#CCCCCC">
       <form action="regpost.jsp" method="post" name="form1">
            <tr align="center">
              <td height="30" colspan="2" bgcolor="#CCCCCC" class="deepred">用户注册</td>
            </tr>
            <tr>
              <td width="50%" height="24" align="center" bgcolor="#FFFFFF">用户名</td>
              <td  align="center"  bgcolor="#FFFFFF"><input  name="username"  type="text"
id="username" size="16"></td>
            </tr>
            <tr>
              <td width="50%" height="24" align="center" bgcolor="#FFFFFF">密码</td>
              <td align="center" bgcolor="#FFFFFF"><input name="password" type="password"
id="password" size="16"></td>
            </tr>
            <tr>
              <td width="50%" height="24" align="center" bgcolor="#FFFFFF">验证密码</td>
              <td align="center" bgcolor="#FFFFFF"><input name="confirm" type="password"
```

```
id="confirm" size="16"></td>
            </tr>
            <tr>
              <td width="50%" height="24" align="center" bgcolor="#FFFFFF">E-mail</td>
              <td  align="center"  bgcolor="#FFFFFF"><input  name="email"  type="text"
id="email" size="16"></td>
            </tr>
            <tr>
              <td  width="50%"  height="24"  align="center"  bgcolor="#FFFFFF"><input
type="submit" name="Submit" value="重写"></td>
              <td align="center" bgcolor="#FFFFFF"><input type="submit" name="Submit2"
value="注册"></td>
            </tr>
      </form>
         </table></td>
        <td width="1" valign="top"></td>
        <td width="100"> </td>
      </tr>
      <tr align="center" bgcolor="green">
        <td height="60" colspan="5" class="white">欢迎来到 Java 世界</td>
      </tr>
    </table>
    </body>
    </html>
```

注册提交页面 regpost.jsp 文件内容如下：

```
    <%@ page contentType="text/html;charset=gb2312" pageEncoding="gb2312" %>
    <%@ page import="beans.reg"%>
    <html>
    <head>
    <title>Java 世界</title>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    </head>
    <body bgcolor="#FFFFFF" leftmargin="0" topmargin="0">
    <%
     String                    username                    =                    new
String(request.getParameter("username").getBytes("ISO8859_1")).trim();
     String                    password                    =                    new
String(request.getParameter("password").getBytes("ISO8859_1")).trim();
     String                    confirm                     =                    new
String(request.getParameter("confirm").getBytes("ISO8859_1")).trim();
     String                     email                      =                    new
String(request.getParameter("email").getBytes("ISO8859_1")).trim();
    %>
    <table width="100%" height="100%" border="0" cellpadding="0" cellspacing="0">
      <tr bgcolor="green">
        <td  height="80"  colspan="5"><table  width="100%"  height="100%"  border="0"
cellpadding="0" cellspacing="0">
          <tr>
            <td width="120"> </td>
            <td class="caption">Java 世界</td>
            <td width="200"> </td>
          </tr>
        </table></td>
      </tr>
      <tr>
        <td width="100" align="center" valign="top"> </td>
        <td width="1" valign="top"></td>
        <td width="400" align="center" valign="top">
    <table width="100%" height="20" border="0" cellpadding="0" cellspacing="0">
          <tr>
            <td> </td>
          </tr>
        </table>
    <jsp:useBean id="regID" class="beans.reg" scope="session"/>
    <%
     if(regID.reg(username,password,confirm,email))
     {
      out.print("ok");
      String newID = regID.getID() + "";
```

```
    %>
        <table    width="100%"    border="0"    cellpadding="0"    cellspacing="1"
bgcolor="#CCCCCC">
          <tr align="center">
            <td height="30" colspan="2" bgcolor="#CCCCCC" class="deepred">恭喜您，注册成
功！</td>
          </tr>
          <tr>
            <td width="50%" height="24" align="center" bgcolor="#FFFFFF">编号</td>
            <td align="center" bgcolor="#FFFFFF"><%=newID%></td>
          </tr>
          <tr>
            <td width="50%" height="24" align="center" bgcolor="#FFFFFF">用户名</td>
            <td align="center" bgcolor="#FFFFFF"><%=username%></td>
          </tr>
          <tr>
            <td width="50%" height="24" align="center" bgcolor="#FFFFFF">密码</td>
            <td align="center" bgcolor="#FFFFFF"><%=password%></td>
          </tr>
          <tr>
            <td width="50%" height="24" align="center" bgcolor="#FFFFFF">E-mail</td>
            <td align="center" bgcolor="#FFFFFF"><%=email%></td>
          </tr>
        </table>
    <%
      out.print("<br>");
      out.print("<a href=javascript:window.close()>关闭</a>");
     }else{
      out.print("注册失败！<br>");
      out.print("该用户名已有人使用，请使用另外的用户名！");
      out.print("<a href=javascript:history.go(-1)>返回</a>");
     }
    %>
       </td>
        <td width="1" valign="top"></td>
        <td width="100"> </td>
      </tr>
      <tr align="center" bgcolor="green">
        <td height="60" colspan="5" class="white">欢迎来到 Java 世界</td>
      </tr>
    </table>
    </body>
    </html>
```

【运行结果】

主页如图 9-19 所示。

注册用户，如图 9-20 所示。

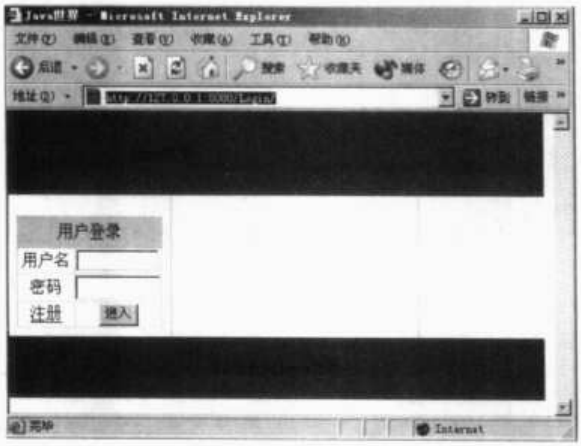


图 9-19



图 9-20

注册信息，如图 9-21 所示。

用户登录（输入正确的用户名和密码后），如图 9-22 所示。

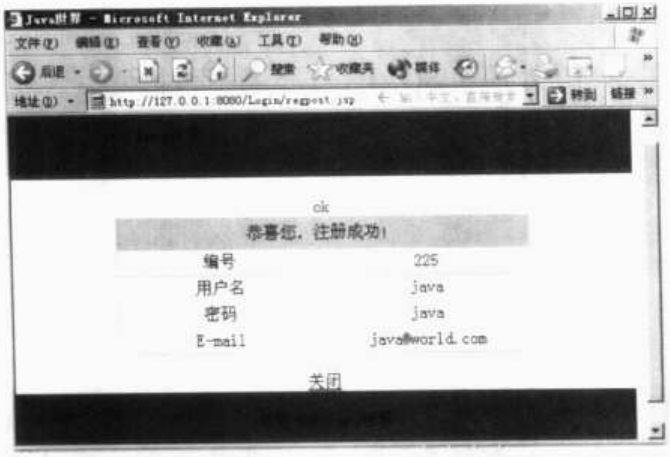


图 9-21

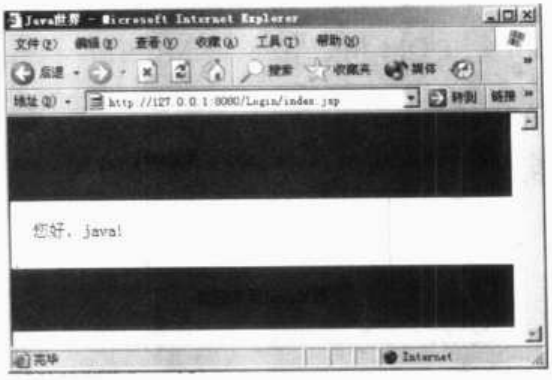


图 9-22

【分析与总结】

本实例是三层体系结构（表现层/事务逻辑层/数据库层的三层体系结构）的典型示例。本例共 7 个文件，index.jsp 先调用 checkLogin，Bean 检查此用户的 Cookie 有无登录记录，没有就显示登录界面，可以单击“注册”按钮进入 reg.jsp 页面，填写完后提交给 regpost.jsp 页面，regpost.jsp 调用 reg，Bean 进行用户的注册或者错误信息返回。登录界面单击“进入”login Servlet。login Servlet 记录了 Cookie 信息后又返回 index.jsp。其中 checkLogin、login、reg 等 Bean 都调用了数据库 Bean DBConn。

小结

本章重点介绍了 JSP 中调用 JavaBean 组件的应用，JavaBean 可以在 JSP 程序中应用的特点使得开发人员可以把某些关键功能和核心算法提取出来，封装成为一个组件对象，增加了代码的重用率、系统的安全性。比如本章中的将访问数据库的功能、数据处理功能编写封装为 JavaBeans 组件，然后在某个 JSP 程序中加以调用。

Java Servlet 是运行在请求/面向请求服务器上的模块。因为 Servlet 使用 Java 语言编写的，所以它是跨平台的。它是一种生成动态内容的解决办法，Java Servlet 可用于处理通过 HTTPS 协议产生的 HTML 表单数据，包括买卖订单或信用卡数据。因此 Servlet 可以成为订单处理系统的一部分，与产品存货数据库一起工作，也可以用于在线支付系统上。

第 10 章　Java 的 Web Service 编程

本章通过 6 个实例简单地介绍了 Java 的 XML 处理和 SOAP 编程。其中包括读写 XML 文件、DOM 树、Web、XML 中的 Server 配置、简单的 SOAP 调用、访问 SOAP RPC 服务和查询 UDDI 信息。可以作为 Web Service 编程的入门。

实例 108　读写 XML 文件

本实例演示如何读写.xml 文件。

【编程要点】

XML 作为全球通用的结构化语言，越来越受人们青睐，各种开发平台（比如 Microsoft Studio 系列、Oracle 系列、Inprise Borland 系列等）也都把支持 XML 开发作为宣传口号之一。在许多项目中利用 XML 数据交换信息，省去了许多麻烦事，不用制定繁琐的数据格式，利用 XML 数据易于表达，也利于一线开发者跟踪调试。

在 XML 应用中，最常用也最实用的莫过于 XML 文件的读写，所以作者通过一个简单的 XML 文件读写来作简要分析。可以在任何文本编辑器中先建立如下结构的 XML 文件，类似于 HTML 结构，但 XML 语义比较严格，起始标记必须配对，比如“〈学生花名册〉”与“〈／学生花名册〉”对应，空格多少可不必在意，但一般都以缩格形式书写，便于阅读。把此文件命名为 Input.xml，可以在任何支持 XML 的浏览器中打开测试一下，如果输入正确，在浏览中可以看到此文件的树形表示结构。

SAX 和 DOM 是用于 XML 解析的最流行的 API，用于 XML 处理的 Java API(JAXP)是一个规范，它为不同销售商创建的 SAX 和 DOM 提供可插接性。DOM（“文档对象模型”）是用与平台和语言无关的方式表示 XML 文档的官方 W3C 标准。DOM 是文档对象模型。XML 将数据组织为一棵树，DOM 就是对这棵树的一个对象描述。通俗地说，就是通过解析 XML 文档，为 XML 文档在逻辑上建立一个树模型，树的节点是一个个对象。我们通过存取这些对象就能够存取 XML 文档的内容。DOM 规范涵盖了文档表示的所有类型的操作，但是它没有涉及例如对文档的语法分析和生成文本输出这样的问题。

具体来说 JAXP 包括了几个工厂类，这就是 JDK1.4 里面的 javax.xml.parsers 包，用来寻找符合 DOM 标准的 XML API 实现类的位置；此外 JAXP 还包括一整套 interface，这就是 JDK1.4 里面的 org.w3c.dom 那几个包。工厂类负责加载 DOM 的实现类。

【程序实现】

Input.xml 的文件内容如下：

```
<?xml version="1.0" encoding="GB2312" ?>
<学生花名册>
  <学生 性别 = "男">
      <姓名>张三</姓名>
      <年龄>20</年龄>
      <电话>1234555</电话>
      </学生>
      <学生 性别 = "男">
      <姓名>李四</姓名>
```

```
    <年龄>19</年龄>
    <电话>8210000</电话>
  </学生>
</学生花名册>
```

StudentBean.java 文件的内容如下：

```
public class StudentBean{
    private String sex;  //学生性别
    private String name; //学生姓名
    private int age;    //学生年龄
    private String phone;  //电话号码

    public void setSex(String s){
       sex  = s;
    }

    public void setName(String s){
     name = s;
    }

    public void setAge(int a){
     age = a;
    }

    public void setPhone(String s){
     phone = s;
    }

    public String getSex(){
     return sex;
    }

    public String getName(){
     return name;
    }

    public int getAge(){
     return age;
    }

    public String getPhone(){
     return phone;
    }
}
```

XMLTest.java 文件内容如下：

```
import java.io.*;                  //Java 基础包，包含各种 I/O 操作
import java.util.*;                //Java 基础包，包含各种标准数据结构操作
import javax.xml.parsers.*;        //XML 解析器接口
import org.w3c.dom.*;              //XML 的 DOM 实现
import org.apache.crimson.tree.XmlDocument; //写 XML 文件要用到

public class XMLTest {
    private Vector student_Vector;

    private void readXMLFile(String inFile) throws Exception {
        //为解析 XML 作准备，创建 DocumentBuilderFactory 实例，指定 DocumentBuilder
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = null;
        try {
            db = dbf.newDocumentBuilder();
        } catch (ParserConfigurationException pce) {
            System.err.println(pce);        //出异常时输出异常信息，然后退出，下同
            System.exit(1);
        }

        Document doc = null;
```

```
        try {
            doc = db.parse(inFile);
        } catch (DOMException dom) {
            System.err.println(dom.getMessage());
            System.exit(1);
        } catch (IOException ioe) {
            System.err.println(ioe);
            System.exit(1);
        }
        //下面是解析 XML 的全过程，比较简单，先取根元素"学生花名册"
        Element root = doc.getDocumentElement();
        //取"学生"元素列表
        NodeList students = root.getElementsByTagName("学生");
        for (int i = 0; i < students.getLength(); i++) {
            //依次取每个"学生"元素
            Element student = (Element) students.item(i);
            //创建一个学生的 Bean 实例
            StudentBean studentBean = new StudentBean();
            //取学生的性别属性
            studentBean.setSex(student.getAttribute("性别"));
            //取"姓名"元素，下面类同
            NodeList names = student.getElementsByTagName("姓名");
            if (names.getLength() == 1) {
                Element e = (Element) names.item(0);
                Text t = (Text) e.getFirstChild();
                studentBean.setName(t.getNodeValue());
            }

            NodeList ages = student.getElementsByTagName("年龄");
            if (ages.getLength() == 1) {
                Element e = (Element) ages.item(0);
                Text t = (Text) e.getFirstChild();
                studentBean.setAge(Integer.parseInt(t.getNodeValue()));
            }

            NodeList phones = student.getElementsByTagName("电话");
            if (phones.getLength() == 1) {
                Element e = (Element) phones.item(0);
                Text t = (Text) e.getFirstChild();
                studentBean.setPhone(t.getNodeValue());
            }
            student_Vector.add(studentBean);
        }
    }

    private void writeXMLFile(String outFile) throws Exception {
        //为解析 XML 作准备，创建 DocumentBuilderFactory 实例，指定 DocumentBuilder
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = null;
        try {
            db = dbf.newDocumentBuilder();
        } catch (ParserConfigurationException pce) {
            System.err.println(pce);
            System.exit(1);
        }

        Document doc = null;
        doc = db.newDocument();

        //下面是建立 XML 文档内容的过程，先建立根元素"学生花名册"
        Element root = doc.createElement("学生花名册");
        //根元素添加上文档
        doc.appendChild(root);

        //取学生信息的 Bean 列表
        for (int i = 0; i < student_Vector.size(); i++) {
```

```
            //依次取每个学生的信息
            StudentBean studentBean = (StudentBean) student_Vector.get(i);
            //建立"学生"元素，添加到根元素
            Element student = doc.createElement("学生");
            student.setAttribute("性别", studentBean.getSex());
            root.appendChild(student);
            //建立"姓名"元素，添加到学生下面，下同
            Element name = doc.createElement("姓名");
            student.appendChild(name);
            Text tName = doc.createTextNode(studentBean.getName());
            name.appendChild(tName);

            Element age = doc.createElement("年龄");
            student.appendChild(age);
            Text tAge =
                doc.createTextNode(String.valueOf(studentBean.getAge()));
            age.appendChild(tAge);
            Element phone = doc.createElement("电话");
            student.appendChild(phone);
            Text tPhone = doc.createTextNode(studentBean.getPhone());
            phone.appendChild(tPhone);
        }
        //把 XML 文档输出到指定的文件
        FileOutputStream outStream = new FileOutputStream(outFile);
        OutputStreamWriter outWriter = new OutputStreamWriter(outStream);
        ((XmlDocument) doc).write(outWriter, "GB2312");
        outWriter.close();
        outStream.close();
    }

    public static void main(String[] args) throws Exception {
        //建立测试实例
        XMLTest xmlTest = new XMLTest();
        //初始化向量列表
        xmlTest.student_Vector = new Vector();

        System.out.println("开始读 Input.xml 文件");
        xmlTest.readXMLFile("Input.xml");

        System.out.println("读入完毕，开始写 Output.xml 文件");
        xmlTest.writeXMLFile("Output.xml");
        System.out.println("写入完成");
    }
}
```

【运行结果】

在当前目录下生成一个内容与 Input.xml 相同的 Output.xml 文件。控制台输出：

```
开始读 Input.xml 文件
读入完毕，开始写 Output.xml 文件
写入完成
```

【分析与总结】

程序中为了保存多个学生信息，借助了一个集合类（并不是单纯意义上的集合，Java 中的集合是集合框架的概念，包含向量、列表、哈希表等），这里采用 Vector 向量类。定义在 XMLTest 测试类中，命名为 student_Vector。然后定义两个方法 readXMLFile 和 writeXMLFile，实现读写操作。

实例 109　DOM 树

本实例演示如何解释 DOM 树并用 Swing 中的 Jtree 控件显示出来。

【编程要点】

XML 将数据组织成一棵树，DOM 是对这棵树的一个对象描述。即通过解析 XML 文档，为其在逻辑上建立一个树模型。树的节点是一个对象，通过存取这些对象可存取 XML 文档的内容。

这里用到的 DOM 分析器 Apache 的 xerces 分析器，必须先下载 Xerces-J-bin.2.5.0 并将其下的包加入到构建路径才行。本程序利用 xerces 分析器得到 DOM Document。

【程序实现】

```
//DOMTreeWalker.java
package webservice;

import org.w3c.dom.*;
import org.w3c.dom.traversal.*;
import org.apache.xerces.parsers.*;
import org.xml.sax.*;
import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.io.*;

public class DOMTreeWalker implements TreeModel {
    TreeWalker walker;

    public DOMTreeWalker(TreeWalker walker) {
        this.walker = walker;
    }

    public DOMTreeWalker(Document document) {
        DocumentTraversal dt = (DocumentTraversal) document;
        walker =
            dt.createTreeWalker(document, NodeFilter.SHOW_ALL, null, false);
    }

    public DOMTreeWalker(Element element) {
        DocumentTraversal dt = (DocumentTraversal) element.getOwnerDocument();
        walker = dt.createTreeWalker(element, NodeFilter.SHOW_ALL, null, false);
    }

    //返回树的根
    public Object getRoot() {
        return walker.getRoot();
    }

    //判断是否为叶子
    public boolean isLeaf(Object node) {
        walker.setCurrentNode((Node) node);
        Node child = walker.firstChild();
        return (child == null);
    }

    //计算该节点的子节点数目
    public int getChildCount(Object node) {
        walker.setCurrentNode((Node) node);
        int numkids = 0;
        Node child = walker.firstChild();
        while (child != null) {
            numkids++;
            child = walker.nextSibling();  //下一个孩子
        }
        return numkids;
    }

    //返回节点的指定子节点
    public Object getChild(Object parent, int index) {
```

```
        walker.setCurrentNode((Node) parent);
        Node child = walker.firstChild();
        while (index-- > 0)
            child = walker.nextSibling();
        return child;
    }

    //返回子节点在父节点当中的位置
    public int getIndexOfChild(Object parent, Object child) {
        walker.setCurrentNode((Node) parent);
        int index = 0;
        Node c = walker.firstChild();
        while ((c != child) && (c != null)) {
            index++;
            c = walker.nextSibling();
        }
        return index;
    }

    //供可修改的树使用的方法，这里没有实现
    public void valueForPathChanged(TreePath path, Object newvalue) {
    }

    //因为这里实现的 DOM Tree 是不可修改的，所以它不触发任何事件
    public void addTreeModelListener(TreeModelListener l) {
    }
    public void removeTreeModelListener(TreeModelListener l) {
    }

    public static void main(String[] args) throws IOException, SAXException {
        //获得用于建立 DOM tree 的 Xerces parser 的对象
        //这里用的是 Apache Xerces APIs
        DOMParser parser = new org.apache.xerces.parsers.DOMParser();
        //获得要分析的 xml 文件
        Reader in = new BufferedReader(new FileReader("web.xml"));
        InputSource input = new org.xml.sax.InputSource(in);
       //开始分析
        parser.parse(input);
        //从分析器中获得 DOM Document
        Document document = parser.getDocument();
        DocumentTraversal traversal = (DocumentTraversal) document;
        //过滤只含有空格的节点
        NodeFilter filter = new NodeFilter() {
            public short acceptNode(Node n) {
                if (n.getNodeType() == Node.TEXT_NODE) {
                    if (((Text) n).getData().trim().length() == 0)
                        return NodeFilter.FILTER_REJECT;
                }
                return NodeFilter.FILTER_ACCEPT;
            }
        };

        //设置显示所有节点但不显示内容
        int whatToShow = NodeFilter.SHOW_ALL & ~NodeFilter.SHOW_COMMENT;
        TreeWalker walker =
            traversal.createTreeWalker(document, whatToShow, filter, false);
        JTree tree = new JTree(new DOMTreeWalker(walker));

        //建立一个窗口和滚动条
        JFrame frame = new JFrame("DOMTreeWalker Demo");
        frame.getContentPane().add(new JScrollPane(tree));
        frame.setSize(500, 250);
        frame.setVisible(true);
    }
}
```

【运行结果】

web.xml 的 DOM Tree 如图 10-1 所示。

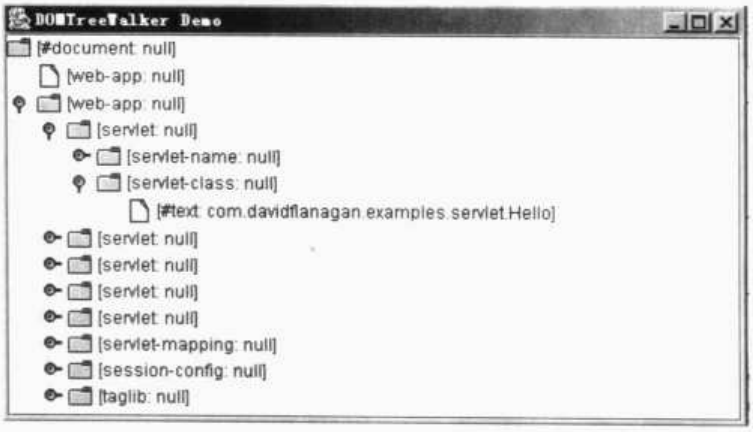


图 10-1

【分析与总结】

TreeWalker 是 Tree 的游标，主要有三个方法，setCurrentNode()用于设置游标从哪个节点开始；firstChild()获得当前节点的第一个孩子的节点，不存在则返回 null；nextSibling()获得下一个孩子节点。通过这三个方法可以用于构造各种方法，如本例中的判断节点是否是叶子 isLeaf()等。

实例 110 web.xml 中的 Servlet 配置

本实例演示如何用 JAXP 的 API 在 web.xml 中添加新的 Servlet。

【编程要点】

JAXP(Java API for XML Parsing)是 Java 应用软件与 XML 的公共接口，用来保证 Java 平台与 XML 的兼容性。它不提供语法分析功能，必须依靠 DOM 和 SAX 等解析 API 才能正常工作。

【程序实现】

```
//WebAppConfig.java
package webservice;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.io.*;

public class WebAppConfig {

    public static void main(String[] args)
        throws IOException, SAXException, ParserConfigurationException {
        WebAppConfig config = new WebAppConfig(new File("web.xml"));

        //添加 servlet name、class mapping 到 DOM tree
        config.addServlet("foo", "bar");
        //输出 DOM tree 的 XML 版本到标准输出口
        config.output(new PrintWriter(new OutputStreamWriter(System.out)));
    }

    org.w3c.dom.Document document;

    //用 JAXP API 来分析 XML 文件
    public WebAppConfig(File configfile)
        throws IOException, SAXException, ParserConfigurationException {
        javax.xml.parsers.DocumentBuilderFactory dbf =
```

```
            DocumentBuilderFactory.newInstance();
        dbf.setValidating(false);
        javax.xml.parsers.DocumentBuilder parser = dbf.newDocumentBuilder();

        //告诉分析器如何处理错误
        parser.setErrorHandler(new org.xml.sax.ErrorHandler() {
            public void warning(SAXParseException e) {
                System.err.println("WARNING: " + e.getMessage());
            }
            public void error(SAXParseException e) {
                System.err.println("ERROR: " + e.getMessage());
            }
            public void fatalError(SAXParseException e) throws SAXException {
                System.err.println("FATAL: " + e.getMessage());
                throw e; //重掷异常
            }
        });
        document = parser.parse(configfile);
    }

    //在 DOM tree 中寻找指定的元素节点，以找到 servlet 名字相应的 class 名字
    public String getServletClass(String servletName) {
        //从所有<servlet>元素中寻找
        NodeList servletnodes = document.getElementsByTagName("servlet");
        int numservlets = servletnodes.getLength();
        for (int i = 0; i < numservlets; i++) {
            Element servletTag = (Element) servletnodes.item(i);
            //获得在<servlet>标记中的<servlet-name>标记
            Element nameTag =
                (Element)
servletTag.getElementsByTagName("servlet-name").item(0);
            if (nameTag == null)
                continue;
            //获得<servlet-name>标记中的 text
            String name = ((Text) nameTag.getFirstChild()).getData().trim();

            //如果<servlet-name> 标记中的 text 与要找的 servlet 相符
            if (servletName.equals(name)) {
                Element classTag =
                    (Element) servletTag.getElementsByTagName(
                        "servlet-class").item(
                        0);
                if (classTag != null) {
                    Text classTagContent = (Text) classTag.getFirstChild();
                    return classTagContent.getNodeValue().trim();
                }
            }
        }
        return null;
    }

    //添加一个新的 name-to-class mapping 到<servlet>标记中
    public void addServlet(String servletName, String className) {
        //建立一个<servlet>标签
        Element newNode = document.createElement("servlet");
        //建立一个<servlet-name> 和 <servlet-class> 标签
        Element nameNode = document.createElement("servlet-name");
        Element classNode = document.createElement("servlet-class");
        //添加 servlet 名和 class 名到标签
        nameNode.appendChild(document.createTextNode(servletName));
        classNode.appendChild(document.createTextNode(className));
        //将这些标签添加到 servlet 标记
        newNode.appendChild(nameNode);
        newNode.appendChild(classNode);

        //建立了新子树后添加到原来的<servlet>标记后
        NodeList servletnodes = document.getElementsByTagName("servlet");
```

```
            Element firstServlet = (Element) servletnodes.item(0);
            firstServlet.getParentNode().insertBefore(newNode, firstServlet);
        }

        //输出 DOM Tree
        public void output(PrintWriter out) {
            XMLDocumentWriter docwriter = new XMLDocumentWriter(out);
            docwriter.write(document);
            docwriter.close();
        }
    }

    //XMLDocumentWriter.java 文件
    package webservice;
    import javax.xml.parsers.*;
    import org.w3c.dom.*;
    import org.xml.sax.*;
    import java.io.*;

    public class WebAppConfig {

        public static void main(String[] args)
            throws IOException, SAXException, ParserConfigurationException {
            WebAppConfig config = new WebAppConfig(new File("web.xml"));

            //添加 servlet name、class mapping 到 DOM tree
            config.addServlet("HelloServlet", "Hello");
            //输出 DOM tree 的 XML 版本到标准输出口
            config.output(new PrintWriter(new OutputStreamWriter(System.out)));
        }

        org.w3c.dom.Document document;

        //用 JAXP API 来分析 XML 文件
        public WebAppConfig(File configfile)
            throws IOException, SAXException, ParserConfigurationException {
            javax.xml.parsers.DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
            dbf.setValidating(false);
            javax.xml.parsers.DocumentBuilder parser = dbf.newDocumentBuilder();

            //告诉分析器如何处理错误
            parser.setErrorHandler(new org.xml.sax.ErrorHandler() {
                public void warning(SAXParseException e) {
                    System.err.println("WARNING: " + e.getMessage());
                }
                public void error(SAXParseException e) {
                    System.err.println("ERROR: " + e.getMessage());
                }
                public void fatalError(SAXParseException e) throws SAXException {
                    System.err.println("FATAL: " + e.getMessage());
                    throw e; //重掷异常
                }
            });
            document = parser.parse(configfile);
        }

        //在 DOM tree 中寻找指定的元素节点，以找到 servlet 名字相应的 class 名字
        public String getServletClass(String servletName) {
            //从所有<servlet>元素中寻找
            NodeList servletnodes = document.getElementsByTagName("servlet");
            int numservlets = servletnodes.getLength();
            for (int i = 0; i < numservlets; i++) {
                Element servletTag = (Element) servletnodes.item(i);
                //获得在<servlet>标记中的<servlet-name>标记
                Element nameTag =
                    (Element)
servletTag.getElementsByTagName("servlet-name").item(0);
```

```
                if (nameTag == null)
                    continue;
                //获得<servlet-name>标记中的text
                String name = ((Text) nameTag.getFirstChild()).getData().trim();

                //如果<servlet-name>标记中的text与要找的servlet相符
                if (servletName.equals(name)) {
                    Element classTag =
                        (Element) servletTag.getElementsByTagName(
                            "servlet-class").item(
                            0);
                    if (classTag != null) {
                        Text classTagContent = (Text) classTag.getFirstChild();
                        return classTagContent.getNodeValue().trim();
                    }
                }
            }
            return null;
        }
        //添加一个新的name-to-class mapping到<servlet>标记中
        public void addServlet(String servletName, String className) {
            //建立一个<servlet>标签
            Element newNode = document.createElement("servlet");
            //建立一个<servlet-name> 和 <servlet-class> 标签
            Element nameNode = document.createElement("servlet-name");
            Element classNode = document.createElement("servlet-class");
            //添加servlet名和class名到标签
            nameNode.appendChild(document.createTextNode(servletName));
            classNode.appendChild(document.createTextNode(className));
            //将这些标签添加到servlet标记
            newNode.appendChild(nameNode);
            newNode.appendChild(classNode);
            //建立了新子树后添加到原来的<servlet>标记后
            NodeList servletnodes = document.getElementsByTagName("servlet");
            Element firstServlet = (Element) servletnodes.item(0);
            firstServlet.getParentNode().insertBefore(newNode, firstServlet);
        }

        //输出DOM Tree
        public void output(PrintWriter out) {
            XMLDocumentWriter docwriter = new XMLDocumentWriter(out);
            docwriter.write(document);
            docwriter.close();
        }
    }
```

【运行结果】

```
<servlet>
   <servlet-name>
      HelloServlet
   </servlet-name>
   <servlet-class>
      Hello
   </servlet-class>
</servlet>
```

添加到了web.xml中，并从控制台输出所有内容。

【分析与总结】

程序的关键仍然是通过SAX parse得到Document，然后再在Document上进行查找和插入Servlet标记的工作。

实例111 简单的SOAP调用

本实例演示一个简单的SOAP调用。

【编程要点】

SOAP 意思是简单对象访问协议（Simple Object Access Protocol）。就如它的名字一样，SOAP 是很简单的。它是一个基于 XML 的协议，允许程序组件和应用程序彼此使用一种标准的 Internet 协议 – HTTP 来通讯。SOAP 是一种独立的平台，它不依赖程序语言，它是简单的、弹性的，很容易扩展的。目前，应用程序能够彼此使用一种基于 DCOM 和 CORBA 技术的远程过程调用（RPC）来进行相互通讯，但 HTTP 不被设计为这个目的。RPC 在 Internet 上应用是非常困难的，它们会出现许多兼容性和安全性的问题，因为防火墙和代理服务器通常都会阻断（block）这些类型的流量。应用程序之间最好的通讯方式是通过 HTTP 协议，因为 HTTP 是支持所有 Internet 浏览器和服务器的。基于这个目的，SOAP 协议被创建出来。目前 Sun One 和.net 架构均完全支持 SOAP。

【程序实现】

服务器端 SimpleMath.java 文件内容如下：

```
package test;
public class SimpleMath {
    public double getSinValue(double input) {
        double ret = Math.sin(input);
        return ret;
    }
}
```

客户机端 SimpleMathClient.java 文件内容如下：

```
package test;
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class SimpleMathClient {

    public static void main(String[] args) throws Exception {
        System.err.println("SOAP call testing");
        double value = Math.random();
        SimpleMathClient smc = new SimpleMathClient();
        //调用远程的 SOAP 服务
        double returnValue = smc.doRequest(value);
        System.err.println("the sin value of " + value + "is: " + returnValue);
    }

    public double doRequest(double value) throws Exception {
        //Build the call.
        Call call = new Call();
        //设置远程对象的 URI
        call.setTargetObjectURI("urn:test.math.sin");
        //设置调用的方法名
        call.setMethodName("getSinValue");
        //设置编码风格
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        //设置方法调用的参数
        Vector params = new Vector();
        params.addElement(
            new Parameter("input", double.class, new Double(value), null));
        call.setParams(params);
        //发送 RPC 请求
        Response resp =
            call.invoke(
                new URL("http://127.0.0.1:8080/soap/servlet/rpcrouter"), "");
        if (resp.generatedFault()) { //远程调用出错处理
```

```
                Fault fault = resp.getFault();
                System.out.println("the call failed: ");
                System.out.println("  Fault Code   = " + fault.getFaultCode());
                System.out.println("  Fault String = " + fault.getFaultString());
                return 0.0d;
            }
            else { //调用成功，获取返回值
                Parameter result = resp.getReturnValue();
                return ((Double) result.getValue()).doubleValue();
            }
        }
    }
```

【运行结果】

首先从 http://ws.apache.org/soap/index.html 下载到 Apache SOAP，这里是 soap-bin-2.3.1.zip，解压后将 webapps 文件夹下的 soap.war 拷贝到 Tomcat 目录的 webapps 文件夹下。然后再将 java mail 下的 mail.jar，用于 xml 分析的 xerces.jar，jaf-1.0.2 下的 activation.jar 及 soap-2_3_1 下的 soap.jar 拷贝到 Tomcat 目录下的 common\lib 文件夹下。将服务器端源文件的 class 文件打包成 test.jar 也放到 Tomcat 的 common\lib 文件夹下。

启动 Tomcat 服务器，输入 URL：http://localhost:8080/soap/admin/，出现如图 10-2 所示的页面则是正确启动了 SOAP。（注：本机 IP 为 127.0.0.1）



图 10-2

进入 Deploy，填写如下信息：

ID：urn:test.math.sin

Scope：Request

Methods：getSinValue

Provider Type：Java

Java Provider Provider Class：test.SimpleMath

Java Provider Static：NO

确认发布，可以单击“List”查看已发布服务。

运行客户机端程序 SimpleMathClient 得到如下结果：

```
SOAP call testing
the sin value of 0.091232755655679 23is: 0.09110624695785023
```

【分析与总结】

从本例中可知道如何发布自己的 SOAP 服务供客户机端调用。

实例 112 访问 SOAP RPC 服务

本实例演示如何利用 HTTP 协议访问 SOAP RPC 服务。

【编程要点】

RPC 与 HTTP 的请求/响应机制的吻合使得 SOAP RPC 常与 HTTP 协议绑定在一起。本实例客户端通过本地服务器向服务器提出请求获取服务器数据，然后通过本地服务器解析后传递给客户端。

【程序实现】

服务器端 StockQuoteService.java 文件内容如下：

```
package webservice;
import java.net.URL;
import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.soap.util.xml.*;

public class StockQuoteService
{
    //建立 DocumentBuilder
    DocumentBuilder xdb = XMLParserUtils.getXMLDocBuilder();
    public Hashtable getInfo(String str) throws Exception
    {
        //建立查询结果集
        Hashtable result=new Hashtable();
        try
        {
            //请求 URL
            URL url =
                new URL(
                    "http://www.xmltoday.com/examples/stockquote"+
                        "/getxmlquote.vep?s="+str);
            //请求回应流
            InputStream is = url.openStream();
            //解析回应的 XML 流
            Document doc = xdb.parse(is);
            //节点合集
            Element element = doc.getDocumentElement();
            //得到 price 标签合集
            NodeList list = element.getElementsByTagName("price");
            //遍历合集
            for(int i=0;i<list.getLength();i++)
            {
                element = (Element) list.item(i);
                String info = element.getAttribute("value");
                switch(i)
                {
                    case 0:
                        result.put("ask",info);
                        break;
                    case 1:
                        result.put("open",info);
                        break;
                    case 2:
                        result.put("dayhigh",info);
                        break;
                    default:
                        result.put("daylow",info);
                }
            }
        }
        catch(Exception e)
```

```
            {
                System.out.println(e.toString());
            }
            //返回解析后的结果合集
            return result;
        }
    }
```

客户机端 StockQuoteClient.java 文件内容如下：

```
package webservice;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class RPCClient
{
    public static void main(String[] args) throws Exception
    {
        //建立远程呼叫
        Call call = new Call();
        //设置目标 URI
        call.setTargetObjectURI("urn:webservice.RPC.Service");
        //设置远程调用方法
        call.setMethodName("getInfo");
        //设置字符编码
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
        //设置远程方法参数
        Vector params = new Vector();
        params.addElement(
            new Parameter("str", String.class, "SUN", null));
        call.setParams(params);
        //设置响应 URL
        Response resp = call.invoke(new URL(
            "http://127.0.0.1:8080/soap/servlet/rpcrouter"),"");
        //得到相应数据
        if (resp.generatedFault())
        {
            Fault fault = resp.getFault();
            System.err.println("Generated fault: " + fault);
        }
        else
        {
            Parameter result = resp.getReturnValue();
            System.out.println(result.getValue());
        }
    }
}
```

【运行结果】

按照上一个例子的方法部署 SOAP，部署成功后运行本实例客户端，就会把 SUN 的股票信息作为一个 Hash 表显示。

【分析与总结】

RPC 还可以与 FTP、SMTP、JMS 等结合，在 Apache SOAP 中还提供了 HTTP 与 SMTP 协议的支持。

实例 113　查询 UDDI 信息

本实例演示查询 IBM 官方 UDDI（Universal Description, Discovery and Integration，通用描述，发现与集成）信息。

【编程要点】

UDDI 是一套基于 Web 分布式的，为 Web Service 提供信息注册的规范并包括一组使得企业可注册自身提供的 Web 服务以使其他企业能够发现的访问协议的实施规范。UDDI 项目鼓励 Web 服务相互操作和相互采用。

它是一种工商界居于领先地位的企业之间的伙伴关系，这种关系最早是由 IBM、Ariba 和 Microsoft 建立的。UDDI 基于现成的标准，如可扩展标记语言（Extensible Markup Language，XML）和简单对象访问协议（Simple Object Access Protocol，SOAP）。UDDI 的所有兼容实现都支持 UDDI 规范。公共规范是机构成员在开放的、兼容并蓄的过程中开发出来的。

UDDI 注册中心包含了通过程序手段可以访问到的对企业和企业支持的服务所做的描述。此外，还包含对 Web 服务所支持的因行业而异的规范、分类法定义（用于对于企业和服务很重要的类别）以及标识系统（用于对于企业很重要的标识）的引用。UDDI 提供了一种编程模型和模式，它定义与注册中心通信的规则。UDDI 规范中所有 API 都用 XML 来定义，包装在 SOAP 信封中，在 HTTP 上传输。

【程序实现】

FindBusinessInfo.java 文件内容如下：

```
package webservice;
import com.ibm.uddi.*;
import com.ibm.uddi.response.*;
import com.ibm.uddi.client.*;
import com.ibm.uddi.datatype.business.*;
import org.w3c.dom.*;
import java.util.*;

public class FindBusinessInfo
{
    public static void main(String[] args)
    {
        FindBusinessInfo demo=new FindBusinessInfo();
        demo.run();
        System.exit(0);
    }

    public void run()
    {
        //建立 UDDI 代理
        UDDIProxy proxy=new UDDIProxy();
        try
        {
            //设置请求页面
            proxy.setInquiryURL
    ("http://www-3.ibm.com/services/uddi/testregistry/inquiryapi");
        }
        catch(Exception e)
        {
            System.out.println(e.toString());
        }
        try
        {
            //获得所有以 A 开头的业务信息
            BusinessList list=proxy.find_business("A",null,0);
            Vector info=
list.getBusinessInfos().getBusinessInfoVector();
            for(int i=0;i<info.size();i++)
            {
                BusinessInfo business=(BusinessInfo)info.elementAt(i);
                System.out.println(business.getNameString());
            }
```

```
            }
            catch(UDDIException ue)
            {
                System.out.println(ue.getDispositionReport());
            }
            catch(Exception e)
            {
                System.out.println(e.toString());
            }
        }
}
```

【分析与总结】

可以查询、添加、删除或修改 UDDI 信息，前提是安装 JSSE（Java Secure Socket Extension）。在 UDDI 注册中心注册一个企业，获取授权的方法和用户密码后才可以修改自己企业的各项信息。

小结

Java 语言是一种可移植的代码，XML 描述的是一种完全可移植的数据，如果二者结合起来能为企业级应用程序的开发提供一种全新的解决方案。本章初步介绍了 JAXP、DOM API 的使用。

本章还初步介绍如何用 SOAP 来调用 Web 服务，SOAP 以 RPC 为一致性的调用途径，使用 HTTP 传送 XML 消息。RPC 提高分布式应用程序开发者的效率，因为开发者无需再费时编写网络底层协议的代码。CORBA、DCOM 和 RMI 都是实现 RPC 的面向对象的机制。

第 11 章　EJB 编程

本章通过 7 个实例介绍了各种 Enterprise JavaBean 的编程，其中包括无状态会话 Bean、有状态会话 Bean、银行存取 BMP、MessageDrivenBean 以及如何在 JSP 调用 EJB。

实例 114　JNDI 的使用

本实例演示如何使用 JNDI。

【编程要点】

Java Naming and Directory Interface（JNDI）是一套提供命名和目录功能的 API，Java 应用程式开发者通过使用 JNDI，在 naming 和 directory 方面的应用上就有了共同的准则。

JNDI 包含一组 API 和一组 SPI（Service Provider Interface）。Java 程式透过 JNDI API 存取各种 naming 和 directory 服务；JNDI SPI 则使得各种 naming 和 directory 服务透明化，允许 Java 程式通过 JNDI API 来存取这些服务。

JNDI 分为三个 packages：

javax.naming 包含 naming 服务的类别（classes）和存取界面（interfaces for accessing）。其中 Context api 让使用者可以定义物件在 namespaces 中的“相对位置”。naming 服务便以 context 为界面，提供查看、系结、物件更名（renaming objects）等功能。InitialContext api 提供 naming 或 directory 服务的一个起始位置。因为在 JNDI 的世界中没有绝对的 root 观念，所有的动作都建立在 context 上；有了起始位置，使用者才能借助它对其 context 上的物件进行存取。

NamingException api 则为 JNDI 定义的一组类别，负责侦测（catch）所有发生在 naming 或 directory 服务里的例外状况（exceptions）。

javax.naming.directory 这个 package 由 javax.naming 扩充而来，提供存取 directory 服务的功能——建立在 naming 服务上，增加对 directory 中的物件检索其属性（retrieve attributes）和透过指定属性为条件来搜寻（search）等功能。DirContext api 提供物件在 directory 内 context 的界面，与 Context api 的运作方式类似，但更进一步定义了查询和更新 directory 中物件属性的方法（methods）。

javax.naming.spi 让系统发展者为特定的 naming 或 directory 系统来撰写使用 JNDI 的应用程式，例如在 Plug-ins、Java Object Support 及 Multiple Naming Systems(Federatio n)等方面的应用。

【程序实现】

JNDIDemo.java 文件内容如下：

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.Hashtable;

public class JNDIDemo {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("格式: java Lookup <filename>");
```

```
            System.exit(-1);
        }
        String name = args[0];
        //指定使用 sun 的文件系统
        Hashtable env = new Hashtable(11);
        env.put(
            Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
        try {
            //初始化 Context,它是连接 JNDI 树的起始点
            Context ctx = new InitialContext(env);
            //查找你要的对象
            Object obj = ctx.lookup(name);
            //打印找到的对象
            System.out.println(name + "绑定在: " + obj);
            //关闭 Context
            ctx.close();
        } catch (NamingException e) {
            System.err.println("Problem looking up " + name + ": " + e);
        }
    }
}
```

【运行结果】

在当前目录输入 java JNDIDemo /tmp

如果 tmp 是文件夹，则显示：

```
/tmp 绑定在: com.sun.jndi.fscontext.RefFSContext@1dae083f
```

如果 tmp 是文件，则显示：

```
/tmp 绑定在: /tmp
```

【分析与总结】

JNDI 作为网络查询的标准方法，用于 EJB、RMI-IIOP 和 JDBC 等领域。

实例 115　构建一个简单的 EJB

本实例构建一个简单的 EJB。

【编程要点】

EJB（Enterprise JavaBeans）是 Sun 公司推出的 J2EE 产品家族的成品，是一个全新的面向服务器端的组件框架，用于开发和部署面向对象的分布式的企业级的 Java 应用。用 EJB 技术编写的应用系统可扩展性好，支持并发性、支持事务处理及多用户、多线程环境下的安全性。EJB 应用程序编写一次，就可以在任何支持 EJB 技术的服务器平台上运行。它提供了比以往的中间层解决方案更好的集成性与互操作性，因此，对于高性能、高可扩展性、高安全性的应用系统，通过使用 EJB 组件技术可以为用户提供更加高效的服务。与其他 J2EE 技术一起，大大增强了 Java 的能力，并推动了 Java 语言在企业级应用系统中的应用。

EJB 技术与其他的 J2EE 技术相互结合，如 JMS、JTS、JNDI、RMI 等技术，为企业级电子商务系统的开发提供了一个安全、可靠、灵活以及伸缩性很强的开发平台。

EJB 技术定义了一组可重用的组件：Enterprise Beans。可以利用这些组件，像搭积木一样的建立分布式应用程序。当代码写好之后，这些组件就被组合到特定的文件中去。每个文件有一个或多个 Enterprise Beans，在加上一些配置参数。最后，这些 Enterprise Beans 被配置到一个装了 EJB 容器的平台上。客户能够通过这些 Beans 的 home 接口，定位到某

个 beans，并产生这个 beans 的一个实例。这样，客户就能够调用 Beans 的应用方法和远程接口。

有两种类型的 Enterprise beans：Session beans 和 entity beans。

Session beans 是一种作为单个的 client 执行的对象。作为对远程的任务请求的响应，容器产生一个 Session beans 的实例。一个 Session beans 有一个 client，从某种程度上来说，一个 Session bean 对于服务器来说就代表了它的那个 client.Session beans 也能用于事务，它能够更新共享的数据，但它不直接描绘这些共享的数据。

Session beans 的生命周期是相对较短的。典型的是，只有当 client 保持会话的时候，Session beans 才是活着的。一旦 client 退出了，Session beans 就不再与 client 相联系了。Session beans 被看成是瞬时的，因为如果容器崩溃了，那么 client 必须重新建立一个新的 Session 对象来继续会话。

一个 Session bean 典型的声明了与 client 的互操作或者会话。也就是说，Session bean 在客户会话期间，通过方法的调用，掌握 Client 的信息。一个具有状态的 Session bean 称为有状态的 Session bean。当 client 终止与 Session beans 互操作的时候，会话终止了，而且 bean 也不再拥有状态值。

一个 Session bean 也可能是一个无状态的 session bean。无状态的 Session beans 并不掌握它的客户的信息或者状态。Client 能够调用 beans 的方法来完成一些操作。但是，beans 只是在方法调用的时候才知道 client 的参数变量。当方法调用完成以后，beans 并不继续保持这些参数变量。

这样，所有的无状态的 session beans 的实例都是相同的，除非它正在方法调用期间。这样，无状态的 Session beans 就能够支持多个 client，容器能够声明一个无状态的 Session beans，能够将任何 Session beans 指定给任何 client。

构造一个 EJB 组件的步骤如下：

（1）编写 Bean 类并编译为.class，包含有 Enterprse Bean 类的公开（Public）商务方法的 Remote 接口、EJB 对象生成库的 Home 接口和实际的 Enterprise Bean 类。

（2）创建部署文件，用 jar 工具将部署文件和 Class 文件生成 EJB-jar 文件。

（3）配置 Bean，将 EJB-jar 文件放到 EJB 容器中。

（4）启动 EJB 服务器，用客户端程序测试。

【程序实现】

HelloWorld.java 的内容如下：

```
package myEjb.Hello;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

//这个接口是客户端与 EJB 对象相互作用的中间途径
public interface HelloWorld extends EJBObject
{
    public String printHelloWorld() throws RemoteException;
}
```

HelloWorldHome.java 的内容如下：

```
package myEjb.Hello;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
/**
```

```
*这是 HelloWorldBean 的 Home 接口，它是 EJB 对象的生成库
*/
public interface HelloWorldHome extends EJBHome
{
   HelloWorld create() throws RemoteException, CreateException;
}
```

HelloWorldBean.java 内容如下：

```
package myEjb.Hello;
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

//无状态会话(将在下一个实例中具体讲解)Bean
public class StringProcessorBean implements SessionBean
{

   //下面是 EJB 必须实现的方法
public void ejbCreate()
{
   System.out.println("ejbCreate()");
}
public void ejbRemove()
{
System.out.println("ejbRemove()");
}
public void ejbActivate()
{
   System.out.println("ejbActivate()");
}

public void ejbPassivate()
{
   System.out.println("ejbPassivate()");
}
public void setSessionContext(SessionContext sc)
{
}

//商务方法 printHelloWorld()
public String printHelloWorld()
{
    return "Hello,World!"
}
}
```

客户端程序 Client.java 的内容如下：

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.Properties;
import com.javapro.ejb.StringProcessor;
import com.javapro.ejb.StringProcessorHome;

//通过 Client 触发调用 Bean 方法
public class Client
{
   public static void main(String[] args)
  (
      //为 JNDI 初始化取得系统属性
      Properties properties = new Properties();
      try {
        //取得初始化属性 jndiContext，它是连接 JNDI 树的起始点
      InitialContext jndiContext = new InitialContext(properties);
        //取得 Home 对象的引用
        Object ref  = jndiContext.lookup("HelloWorldHome");
        HelloWorldHome home = (HelloWorldHome)
      PortableRemoteObject.narrow (ref, HelloWorldHome.class);
        //用 EJB 生成库 Home 生成 EJB 对象
        HelloWorld hello= home.create();
```

```
            //调用 EJB 对象方法 printHelloWorld ( ) ,EJB 对象把调用委派给 Bean
            System.out.println (hello.printHelloWorld());
        }
        catch(Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

Ejb-jar.xml 部署文件内容如下：

```
<!DOCTYPE ejb-jar>
<ejb-jar>
    <description>Simple EJB application </description>
    <display-name>HelloWorld Application</display-name>
    <enterprise-beans>
     <session>
      <ejb-name>HelloWorld</ejb-name>
        <home>Hello.HelloWorldHome</home>
        <remote>Hello.HelloWorld</remote>
        <ejb-class>Hello.HelloWorldBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Bean</transaction-type>
       </session>
      </enterprise-beans>
</ejb-jar>
```

【运行结果】

安装 Jboss 作为 EJB 的服务器，首先编译 HelloWorld.java、HelloWorldHome.java、HelloWorldBean.java；然后建立部署文件，把.class 文件和 ejb-jar.xml 放在工程的 MATA-INF 目录下（没有的话要自己创建），部署文件打成 JAR 包，放到 Jboss 的 server\default\deploy 目录下。

启动 jboss 后，classpath 中加入%JBOSSHOME%\client\lib 中的 jbossall-client.jar、jnet.jar、log4j.jar，编译、运行 Client 程序，控制台输出 Hello,World!

【分析与总结】

许多复杂的 EJB 均可以本实例为模板。

实例 116　无状态会话 Bean

本实例创建一个无状态的 session Bean。

【编程要点】

无状态（stateless）Bean 的 ejbCreate()方法不能带任何参数，并且一次对话只持续单一的方法调用，不保存对话状态。在无状态的 Bean 部署文件中，Session-type 为 stateless。

本程序用 Eclipse+Lomboz+Jboss 来完成，它们的安装和配置在最后一章有专门介绍。其中 CalculateBean.java 和 Client 需要我们修改，其他的文件可以由 Lomboz 自动生成。步骤如下：

首先，打开“File”→“New”→“Lomboz J2EE Project”，创建一个名为 CalculateEJB 的 J2EE Project。单击“Next”，默认各种设置，再单击“Next”，进入下一个对话框，在 Targeted Server 中加入已经配置好的服务器，如图 11-1 所示。

在 EJB Modules 中新建 EJB 模块，在本例中，命名为 Calculate，然后单击“Finish”。这样，一个 J2EE 项目的框架就自动生成了。

然后通过 Lomboz EJB Creation Wizard 向 MyEJB 模块中添加一个 Session Bean。单击“File”→“New”→“Lomboz EJB Creation Wizard”，或者右击 Calculate 模块，选择“New”

→ “Lomboz EJB Creation Wizard”，显示出如图 11-2 所示对话框。



图 11-1

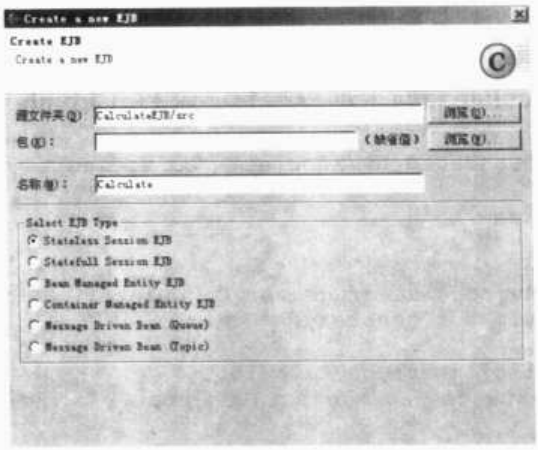


图 11-2

在 cal 这个包下面创建一个名为 Calculate 的 Stateless Session Bean。完成以后，在 CalculateEJB 项目的 src 文件夹下面，出现了一个名为 CalculateBean.java 的文件，右击这个文件，选择“New”→“Lomboz EJB Method Wizard”，可以给这个 EJB 添加各种方法。然后，需要把 Calculate 这个 Session Bean 添加到 Calculate 这个模块上面：右击 CalculateBean.java，选择“Lomboz J2EE”→“Add EJB to Module”，再选择 Calculate 即可。

右击 MyEJB 模块，选择“LombozJ2EE”→“Generate EJB Classes”，这样就在 ejbsrc 目录下面自动生成了下面几个文件：

CalculateHome.java

CalculateHome.java

CalculateLocalHome.java

CalculateLocal.java

CalculateSession.java (CalculateSession extends CalculateBean)

CalculateUtil.java

Build project 以后，自动将模块打包完毕。右击 Calculate 文件夹，选择“Lomboz J2EE”→“Deploy Model”将 jar 包部署到 jboss 上。最后，建立一个应用程序 Client.java 来测试 EJB。通过“File”→“New”→“Lomboz EJB Test Client Wizard”打开一个对话框，按图 11-3 填写。



图 11-3

完成以后，在 src 目录下面生成了 Client.java，在其中的 testBean 方法中加入测试代码即可。

【程序实现】

Calculate.java（远程接口）文件（Lomboz 自动产生）如下：

```
/*
 * Generated by XDoclet - Do not edit!
 */
package cal;

/**
 * Remote interface for Calculate.
 * @lomboz generated
 */
public interface Calculate
   extends javax.ejb.EJBObject
{
   public int add( int a,int b )
      throws java.rmi.RemoteException;
}
```

CalculateHome.java（Home）文件（Lomboz 自动生成）内容如下：

```
/*
 * Generated by XDoclet - Do not edit!
 */
package cal;

/**
 * Home interface for Calculate.
 * @lomboz generated
 */
public interface CalculateHome
   extends javax.ejb.EJBHome
{
   public static final String COMP_NAME="java:comp/env/ejb/Calculate";
   public static final String JNDI_NAME="CalculateBean";

   public cal.Calculate create()
      throws javax.ejb.CreateException,java.rmi.RemoteException;

}
```

CalculateBean.java 文件内容如下：

```
package cal;

import javax.ejb.SessionBean;

/**
 * @ejb.bean name="Calculate"
 *   jndi-name="CalculateBean"
 *   type="Stateless"
 *
 *--
 * This is needed for JOnAS.
 * If you are not using JOnAS you can safely remove the tags below.
 * @jonas.bean ejb-name="Calculate"
 *   jndi-name="CalculateBean"
 *
 *--
 **/

public abstract class CalculateBean implements SessionBean {
     /**
      * @ejb.interface-method
      *   view-type="remote"
     **/
    //添加的方法
     public int add(int a, int b) {
          return a + b;
```

```
        }
    }
```

ejb-jar.xml 文件内容（Lomboz 自动生成）如下：

```
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">
    <ejb-jar >
      <description><![CDATA[No Description.]]></description>
      <display-name>Generated by XDoclet</display-name>
      <enterprise-beans>
        <!-- Session Beans -->
        <session >
          <description><![CDATA[]]></description>
          <ejb-name>Calculate</ejb-name>
          <home>cal.CalculateHome</home>
          <remote>cal.Calculate</remote>
          <local-home>cal.CalculateLocalHome</local-home>
          <local>cal.CalculateLocal</local>
          <ejb-class>cal.CalculateSession</ejb-class>
          <session-type>Stateless</session-type>
          <transaction-type>Container</transaction-type>
        </session>
        <!--
          To add session beans that you have deployment descriptor info for, add
          a file to your XDoclet merge directory called session-beans.xml that contains
          the <session></session> markup for those beans.
        -->
        <!-- Entity Beans -->
        <!--
          To add entity beans that you have deployment descriptor info for, add
          a file to your XDoclet merge directory called entity-beans.xml that contains
          the <entity></entity> markup for those beans.
        -->
        <!-- Message Driven Beans -->
        <!--
          To add message driven beans that you have deployment descriptor info for, add
          a file to your XDoclet merge directory called message-driven-beans.xml that
contains
          the <message-driven></message-driven> markup for those beans.
        -->
      </enterprise-beans>
      <!-- Relationships -->
      <!-- Assembly Descriptor -->
      <assembly-descriptor >
        <!--
          To add additional assembly descriptor info here, add a file to your
          XDoclet merge directory called assembly-descriptor.xml that contains
          the <assembly-descriptor></assembly-descriptor> markup.
        -->
      <!-- finder permissions -->
      <!-- transactions -->
      <!-- finder transactions -->
      </assembly-descriptor>
    </ejb-jar>
```

Client.java（客户机程序）文件内容：

```
    package cal;

    import java.rmi.RemoteException;
    import java.util.Hashtable;
    import javax.ejb.CreateException;
    import javax.naming.InitialContext;
    import javax.naming.NamingException;

    public class Client {

        private cal.CalculateHome getHome() throws NamingException {
            return (cal.CalculateHome) getContext().lookup(
                cal.CalculateHome.JNDI_NAME);
        }
```

```
    private InitialContext getContext() throws NamingException {
        Hashtable props = new Hashtable();

        props.put(
            InitialContext.INITIAL_CONTEXT_FACTORY,
            "org.jnp.interfaces.NamingContextFactory");
        props.put(InitialContext.PROVIDER_URL, "jnp://127.0.0.1:1099");

        //This establishes the security for authorization/authentication
        //props.put(InitialContext.SECURITY_PRINCIPAL,"username");
        //props.put(InitialContext.SECURITY_CREDENTIALS,"password");

        InitialContext initialContext = new InitialContext(props);
        return initialContext;
    }
    public void testBean() {

        try {
            cal.Calculate myBean = getHome().create();

            //-------------------------------------
            //This is the place you make your calls.
            //System.out.println(myBean.callYourMethod());
            System.out.println("5+8 = ");
            System.out.println(myBean.add(5, 8));

        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (CreateException e) {
            e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Client test = new Client();
        test.testBean();
    }
}
```

【运行结果】

控制台输出 5+8=13。

【分析与总结】

无状态 session Bean 不保存任何对话状态，所以消耗的资源也比较少，可以应用于高性能引擎的数学计算。

实例 117 简单的有状态会话 Bean

本实例利用有状态的 session Bean 实现一个简单的累加器，当前的计数值保存在对应的 Bean 中，当客户端请求时加 1。

【编程要点】

有状态 session Bean 在对 Bean 的不同调用期间保留有用的状态。在编写 Bean 之前必须了解状态会话 Bean 的特征，如必须实现 SessionBean 接口。一个客户端程序调用状态 Bean 中的方法时，客户端开始与 Bean 对话，每个 Bean 均保存特定客户的状态。还需要了解激活/钝化等概念。

利用 Lomboz J2EE Wizards 创建一个 statefull session Bean，再往 Bean 中添加两个方法：bussiness method: count 和 create method: ejbCreate 方法。生成 EJB 文件后部署到 jboss，然后用向导添加 CountClient.java 测试程序。

【程序实现】

CountBean.java 文件的内容如下：

```
/*
 * 创建日期 2004-10-15
 *
 * 更改所生成文件模板为
 * 窗口 > 首选项 > Java > 代码生成 > 代码和注释
 */
package count;

import javax.ejb.SessionBean;
import javax.ejb.*;

/**
 * @ejb.bean name="Count"
 *   jndi-name="CountBean"
 *   type="Stateful"
 *
 *--
 * This is needed for JOnAS.
 * If you are not using JOnAS you can safely remove the tags below.
 * @jonas.bean ejb-name="Count"
 *   jndi-name="CountBean"
 *
 *--
 **/

public abstract class CountBean implements SessionBean {
    int val;  //这个计数值是有会话状态的
    /**
     * @ejb.interface-method
     *   view-type="remote"
    **/
    public int count() {
        return ++val;
    }
    /**
     * @ejb.create-method
     *   view-type="remote"
    **/
    public void ejbCreate(int value) throws CreateException {
        this.val = value;
    }
}
```

CountClient.java（测试程序）文件内容如下：

```
/*
 * 创建日期 2004-10-15
 *
 * 更改所生成文件模板为
 * 窗口 > 首选项 > Java > 代码生成 > 代码和注释
 */
package count;

import java.rmi.RemoteException;
import java.util.Hashtable;

import javax.ejb.CreateException;
import javax.naming.InitialContext;
import javax.naming.NamingException;

/**
 * @author zhqq
 *
 * 更改所生成类型注释的模板为
 * 窗口 > 首选项 > Java > 代码生成 > 代码和注释
 */
```

```
public class CountClient {
    private count.CountHome getHome() throws NamingException {
        return (count.CountHome) getContext().lookup(count.CountHome.JNDI_NAME);
    }
    private InitialContext getContext() throws NamingException {
        Hashtable props = new Hashtable();

        props.put(
            InitialContext.INITIAL_CONTEXT_FACTORY,
            "org.jnp.interfaces.NamingContextFactory");
        props.put(InitialContext.PROVIDER_URL, "jnp://127.0.0.1:1099");

        //This establishes the security for authorization/authentication
        //props.put(InitialContext.SECURITY_PRINCIPAL,"username");
        //props.put(InitialContext.SECURITY_CREDENTIALS,"password");

        InitialContext initialContext = new InitialContext(props);
        return initialContext;
    }
    public void testBean() {

        try {
            //count.Count myBean = getHome().create();

            //-------------------------------------
            //This is the place you make your calls.
            //System.out.println(myBean.callYourMethod());

            //具有 3 个 Count EJB 对象的数组
            Count count[] = new Count[3];
            int countVal = 0;
            System.out.println("Instantiating beans . . . ");
            for (int i = 0; i < 3; i++) {
            //创建 EJB 对象并将当前的计数器初始化
                count[i] = getHome().create(countVal);
                countVal = count[i].count();
                System.out.println(countVal);
                try{
                    Thread.sleep(500);
                }
                catch(Exception e){
                    System.out.println(e.toString());
                }
            }
            //调用每一个 EJB 对象的 count()方法，保证 Bean 正常被激活和钝化
            System.out.println("Calling count() on beans . . . ");
            for (int i = 0; i < 3; i++) {
                countVal = count[i].count();
                System.out.println(countVal);
                try{
                    Thread.sleep(500);
                }
                catch(Exception e){
                    System.out.println(e.toString());
                }
            }
            for (int i = 0; i < 3; i++) {
                //EJB 对象使用完毕，从内存中清除
                try{
                    count[i].remove();
                }
                catch(Exception e){System.out.println(e.toString());}
            }

        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (CreateException e) {
            e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
```

```
        }
    }

    public static void main(String[] args) {
        CountClient test = new CountClient();
        test.testBean();

    }
}
```

【运行结果】

```
Instantiating beans . . .
1
2
3
Calling count() on beans . . .
2
3
4
```

【分析与总结】

很多商务过程是一个多请求且持续较久的对话，所以有状态会话可以应用在许多领域中。如网上购物，用户可以往购物车添加物品。

实例 118　银行存取 BMP

本实例编写一个用于银行存取的 BMP。

【编程要点】

Entity Beans 对数据库中的数据提供了一种对象的视图。例如：一个 Entity bean 能够模拟数据库表中一行相关的数据。多个 client 能够共享访问同一个 Entity bean，多个 client 也能够同时的访问同一个 Entity bean；Entity beans 通过事务的上下文来访问或更新下层的数据。这样，数据的完整性就能够被保证。

Entity Beans 能存活相对较长的时间，并且状态是持续的。只要数据库中的数据存在，Entity beans 就一直存活。而不是按照应用程序或者服务进程来说的。即使 EJB 容器崩溃了，Entity beans 也是存活的。Entity Beans 生命周期能够被容器或者 Beans 自己管理。如果由容器控制着保证 Entity beans 持续的 issus。如果由 Beans 自己管理，就必须写 Entity beans 的代码，包括访问数据库的调用。

Entity Beans 是由主键（primary key 一种惟一的对象标识符）标识的。通常，主键与标识数据库中的一块数据（例如一个表中的一行的主键）是相同的。主键是 client 能够定位特定的数据块。

BMP 是 Bean 管理持久的简称，它是 Entity Bean 实现持久化的方法之一。对于所有实体，Bean 必须实现 javax.ejb.EntityBean 接口，该接口定义 Bean 应实现的调用返回方法，还可以定义其他方法。在 BMP 中必须定义 ejbFind()方法，并且必须有 ejbFindPrimaryKey()定位器。在本实例的 ejbCreate()中，Home 对象返回一个 EJB 对象给客户端，而 Entity Bean 实例返回一个主键给 EJB 容器。

本实例用到了 SQLServer 的数据库，先创建 bank 数据库，它包含有一个表 accounts，有三个列属性：id(varchar)、ownerName(varchar)、banlance(decemial)，并保存有一个记录。

【程序实现】

AccountPK.java（主键定义类）文件内容如下：

```
package account;
```

```
import java.io.Serializable;
//账户的主键类
public class AccountPK implements java.io.Serializable {
    public String accountID;
    public AccountPK(String id) {
        this.accountID = id;
    }
    public AccountPK() {
    }
    //返回被存储的域
    public String toString() {
        return accountID;
    }
    //主键存储在一个 Hash 表中
    public int hashCode() {
        return accountID.hashCode();
    }
    //比较两个主键，来决定实体 Bean 是否代表同一个数据库数据
    public boolean equals(Object account) {
        return ((AccountPK) account).accountID.equals(accountID);
    }
}
```

Account.java（远程接口）文件内容如下：

```
package account;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Account extends EJBObject {
    //存 amt 到账户中去。
    public void deposit(double amt) throws AccountException, RemoteException;
    //从账户中取出 amt，如果 amt>账户余额抛出异常
    public void withdraw(double amt) throws AccountException, RemoteException;
    //在实体 Bean 上的获得器/设置器方法
    public double getBalance() throws RemoteException;
    public String getOwnerName() throws RemoteException;
    public void setOwnerName(String name) throws RemoteException;
    public String getAccountID() throws RemoteException;
    public void setAccountID(String id) throws RemoteException;
}
```

AccountHome.java（Home 接口）文件内容如下：

```
package account;
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.util.Collection;
import java.rmi.RemoteException;
public interface AccountHome extends EJBHome {
    //我们定义了单独的 create()方法在这个本地接口中
    //它对应 AccountBean 中的 ejbCreate()方法
    Account create(String accountID, String ownerName)
        throws CreateException, RemoteException;
    //通过主键(accountID)查找一个账户
    public Account findByPrimaryKey(AccountPK key)
        throws FinderException, RemoteException;
    //用它的所有者的名字查找账户
    public Collection findByOwnerName(String name)
        throws FinderException, RemoteException;
    //返回所有账户
    public double getTotalBankValue() throws AccountException, RemoteException;
}
```

AccountBean（实体 Bean 类）文件内容如下：

```
package account;
import java.sql.*;
import javax.ejb.*;
import java.util.*;
import javax.naming.*;
```

```
//这个管理持久化实体Bean表示一个银行账户
public class AccountBean implements EntityBean {
    protected EntityContext ctx;
    //Bean的状态域
    private String accountID; //本例中的主键
    private String ownerName;
    private double balance;
    public AccountBean() {
    }
    //商务逻辑方法
    //存款
    public void deposit(double amt) throws AccountException {
        balance += amt;
    }
    //取款，如果amt>余额，抛出异常
    public void withdraw(double amt) throws AccountException {
        if (amt > balance) {
            throw new AccountException("余额不够");
        }
        balance -= amt;
    }
    //实体Bean上的获得器/设置器方法
    public double getBalance() {
        System.out.println("getBalance()");
        return balance;
    }
    public void setOwnerName(String name) {
        System.out.println("setOwnerName()" + accountID);
        ownerName = name;
    }
    public String getOwnerName() {
        System.out.println("getOwnerName() on" + accountID);
        return ownerName;
    }
    public String getAccountID() {
        System.out.println("getAccountID() on" + accountID);
        return accountID;
    }
    public void setAccountID(String id) {
        System.out.println("setAccountID() on " + accountID);
        this.accountID = id;
    }
    //返回所有账户的余额
    public double ejbHomeGetTotalBankValue() throws AccountException {
        PreparedStatement pstmt = null;
        Connection conn = null;
        try {
            //和数据库连接
            conn = getConnection();
            //执行查询操作，得到所有账户余额总和
            pstmt =
                conn.prepareStatement(
                    "select sum(balance) as total from accounts");
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                return rs.getDouble("total");
            }
        } catch (Exception e) {
            e.printStackTrace();
            throw new AccountException(e);
        } finally { //释放数据库资源
            try {
                if (pstmt != null)
                    pstmt.close();
            } catch (Exception e) {
            }
```

```
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
            }
        }
        throw new AccountException("Error!");
    }
    //EJB 必须实现的方法
    public void ejbActivate() {
        System.out.println("ejbActivate()");
    }
    //从数据库中移除实体 Bean
    public void ejbRemove() throws RemoveException {
        System.out.println("ejbRemove()");
        //一个实体 Bean 可以表示不同的数据实例
        //我们应该通过主键来判断删除哪个数据实例
        AccountPK pk = (AccountPK) ctx.getPrimaryKey();
        String id = pk.accountID;
        PreparedStatement pstmt = null;
        Connection conn = null;
        try {
            //获得新的数据库连接
            conn = getConnection();
            //从数据库中删除账号
            pstmt = conn.prepareStatement("delete from accounts where id = ?");
            pstmt.setString(1, id);
            //操作数据库失败，抛出一个系统级错误
            if (pstmt.executeUpdate() == 0) {
                throw new RemoveException(
                    "Account " + pk + " failed to be removed from the database");
            }
        } catch (Exception ex) {
            throw new EJBException(ex.toString());
        } finally { //释放数据库连接
            try {
                if (pstmt != null)
                    pstmt.close();
            } catch (Exception e) {
            }
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
            }
        }
    }
    //钝化，为了挂起释放的资源
    public void ejPassivate() {
        System.out.println("ejbPassivate ()on " + accountID);
    }
    //更新内存中的实体 Bean 对象，反映数据库中的当前值
    public void ejbLoad() {
        System.out.println("ejbLoad() on " + accountID);
        //得到实体主键，这样我们知道哪个实例被加载
        AccountPK pk = (AccountPK) ctx.getPrimaryKey();
        String id = pk.accountID;
        PreparedStatement pstmt = null;
        Connection conn = null;
        try {
            //获得新的数据库连接
            conn = getConnection();
            //通过 ID 查询，得到数据库中的账号
            pstmt =
                conn.prepareStatement(
                    "select ownerName, balance from accounts where id = ?");
            pstmt.setString(1, id);
```

```
            ResultSet rs = pstmt.executeQuery();
            rs.next();
            ownerName = rs.getString("ownerName");
            balance = rs.getDouble("balance");
        } catch (Exception ex) {
            throw new EJBException(
                "Account " + pk + " failed to load from database",ex);
        } finally { //释放数据库连接
            try {
                if (pstmt != null)
                    pstmt.close();
            } catch (Exception e) {
            }
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
            }
        }
    }
    //更新数据库，反映当前内存中的值
    public void ejbStore() {
        System.out.println("ejbStore() on " + accountID);
        PreparedStatement pstmt = null;
        Connection conn = null;
        try {
            //获得新的数据库连接
            conn = getConnection();
            //往数据库存储账户
            pstmt =
                conn.prepareStatement(
                    "update accounts set ownerName = ?, balance = ? where id = ?");
            pstmt.setString(1, ownerName);
            pstmt.setDouble(2, balance);
            pstmt.setString(3, accountID);
            pstmt.executeUpdate();
        } catch (Exception ex) {
            throw new EJBException(
                "Account " + accountID + " failed to save to database",
                ex);
        } finally { //释放数据库连接
            try {
                if (pstmt != null)
                    pstmt.close();
            } catch (Exception e) {
            }
            try {
                if (conn != null)
                    conn.close();
            } catch (Exception e) {
            }
        }
    }
    //把一个 Bean 实例和一个特定的环境对象关联起来
    public void setEntityContext(EntityContext ctx) {
        System.out.println("setEntityContext called");
        this.ctx = ctx;
    }
    //把一个 Bean 和它的环境变量分离
    public void unsetEntityContext() {
        System.out.println("unsetEntityContext called on " + accountID);
        this.ctx = null;
    }
    //在 ejbCreate()调用后，Bean 可以从它的环境中提取 EJB 对象
    //并当作一个 this 参数传递
    public void ejbPostCreate(String accountID, String ownerName) {
    }
    //此方法对应 Home 中的 create()方法
```

```
        //当客户端调用本地对象(Home)的 create()方法时,
        //本地对象调用 ejbCreate()方法
        public AccountPK ejbCreate(String accountID, String ownerName)
            throws CreateException {
            PreparedStatement pstmt = null;
            Connection conn = null;
            try {
                System.out.println("ejbCreate() called on " + accountID);
                this.accountID = accountID;
                this.ownerName = ownerName;
                this.balance = 0;
                //获得数据库的连接
                conn = getConnection();
                //插入账号
                pstmt =
                    conn.prepareStatement(
                        "insert into accounts (id, ownerName, balance) values
(?, ?, ?)");
                pstmt.setString(1, accountID);
                pstmt.setString(2, ownerName);
                pstmt.setDouble(3, balance);
                pstmt.executeUpdate();
                //产生一个主键,并返回
                return new AccountPK(accountID);
            } catch (Exception e) {
                throw new CreateException(e.toString());
            } finally { //释放数据库的连接
                try {
                    if (pstmt != null)
                        pstmt.close();
                } catch (Exception e) {
                }
                try {
                    if (conn != null)
                        conn.close();
                } catch (Exception e) {
                }
            }
        }
        //通过主键找到一个账号
        public AccountPK ejbFindByPrimaryKey(AccountPK key)
            throws FinderException {
            PreparedStatement pstmt = null;
            Connection conn = null;
            try {
                System.out.println("ejbFindByPrimaryKey(" + key + ") called");
                //数据库连接
                conn = getConnection();
                //在数据库中查找这个实体
                pstmt =
                    conn.prepareStatement("select id from accounts where id = ?");
                pstmt.setString(1, key.toString());
                ResultSet rs = pstmt.executeQuery();
                rs.next();
                //返回主键
                return key;
            } catch (Exception e) {
                throw new FinderException(e.toString());
            } finally { //释放数据库
                try {
                    if (pstmt != null)
                        pstmt.close();
                } catch (Exception e) {
                }
                try {
                    if (conn != null)
                        conn.close();
                } catch (Exception e) {
```

```
                }
            }
        }
        //通过名字查找所有账号
        public Collection ejbFindByOwnerName(String name) throws FinderException {
            PreparedStatement pstmt = null;
            Connection conn = null;
            Vector v = new Vector();
            try {
                System.out.println("ejbFindByOwnerName(" + name + ") called");
                //获得数据库连接
                conn = getConnection();
                //在数据库中找这个主键
                pstmt =
                    conn.prepareStatement(
                        "select id from accounts where ownerName = ?");
                pstmt.setString(1, name);
                ResultSet rs = pstmt.executeQuery();
                //把找到的主键放到 Vector 中
                while (rs.next()) {
                    String id = rs.getString("id");
                    v.addElement(new AccountPK(id));
                }
                //返回一个找到的主键的列举
                return v;
            } catch (Exception e) {
                throw new FinderException(e.toString());
            } finally { //释放数据库
                try {
                    if (pstmt != null)
                        pstmt.close();
                } catch (Exception e) {
                }
                try {
                    if (conn != null)
                        conn.close();
                } catch (Exception e) {
                }
            }
        }

        //返回 JDBC 连接
        public Connection getConnection() throws Exception {
            try {
                Context ctx = new InitialContext();
                //用 jndi name 来查找数据源
                javax.sql.DataSource ds =
                    (javax.sql.DataSource) ctx.lookup("java:/MSSQLDS");
                return ds.getConnection();
            } catch (Exception e) {
                System.out.println("Couldn't get dataSource!");
                e.printStackTrace();
                throw e;
            }
        }
    }
```

ejb-jar.xml（部署文件）的内容如下：

```
    <?xml version="1.0"?>
    <!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
    <ejb-jar>
     <enterprise-beans>
      <entity>
       <ejb-name>Account</ejb-name>
       <home>account.AccountHome</home>
       <remote>account.Account</remote>
       <ejb-class>account.AccountBean</ejb-class>
       <persistence-type>Bean</persistence-type>
```

```
  <prim-key-class>account.AccountPK</prim-key-class>
  <reentrant>False</reentrant>
 </entity>
</enterprise-beans>
<assembly-descriptor>
 <container-transaction>
  <method>
   <ejb-name>Account</ejb-name>
   <method-intf>Remote</method-intf>
   <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
 </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

AccountClient.java（客户端测试程序）文件内容如下：

```
package account;

import javax.naming.*;
import java.util.*;

public class AccountClient {
    public static void main(String[] args) throws Exception {
        Account account = null;
        try { //获取一个本地对象的引用
            Properties p = new Properties();
            p.setProperty(
                "java.naming.factory.initial",
                "org.jnp.interfaces.NamingContextFactory");
            p.setProperty("java.naming.provider.url", "localhost:1099");
            //获得名称上下文空间
            InitialContext jndiContext = new InitialContext(p);
            System.out.println("Got context");
            //jboss 默认 jndi 名为 ejb-jar.xml 中的:ejb-name
            Context ctx = new InitialContext(p);
            Object obj = ctx.lookup("Account");
            AccountHome home = (AccountHome) obj;
            System.err.println(
                "Total of all accounts in bank initially = "
                    + home.getTotalBankValue());
            //生成 EJB 对象
            home.create("123456789", "Kate Green");
            //查找一个账户
            Iterator i = home.findByOwnerName("Kate Green").iterator();
            if (i.hasNext()) {
                account = (Account) i.next();
                javax.rmi.PortableRemoteObject.narrow(account, Account.class);
            } else {
                throw new Exception("Could not find account");
            }
            //调用 balance()方法，并打印输出
            System.out.println("Initial Balance = " + account.getBalance());
            //存款 100
            account.deposit(100);
            System.out.println(
                "After depositing 100, account balance = "+ account.getBalance());
            System.out.println(
                "Total of all accounts in bank now = "+ home.getTotalBankValue());
            //提取 EJB 对象的主键
            AccountPK pk = (AccountPK) account.getPrimaryKey();
            //释放我们的老的 EJB 对象的引用，通过 ID 主键号查找
            account = null;
            account = home.findByPrimaryKey(pk);
            //打印现有的余额
            System.out.println(
                "Found account with ID "
                    + pk
```

```
                        + " Balance = "
                        + account.getBalance());
                //取款 150
                System.out.println(
                    "Now trying to withdraw $150, which is more "
                        + "than is currently available.  This should "
                        + "generate an exception..");
                account.withdraw(150);
            } catch (Exception e) {
                System.out.println("Caught exception!\n" + e);
                //e.printStackTrace();
            } finally { //永久的删除实体
                try {
                    System.out.println("Destroying account..");
                    if (account != null) {
                        account.remove();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

【运行结果】

先配置 Jboss 连接 SQLServer 数据库，将 JDBC 的 lib 文件夹中的 jar 文件拷贝到 Jboss 下的 server\default\lib 下，然后将 Jboss 目录下 docs\examples\jca 下的 mssql-service.xml 文件拷贝到 server\default\deploy 下，然后在以下部分修改：

```
<attribute name="JndiName">MSSQLDS</attribute>
<attribute name="ManagedConnectionFactoryProperties">
<properties>
<config-property name="ConnectionURL"type="java.lang.String">
jdbc:microsoft:sqlserver://192.168.29.126:1433;DatabaseName=bank
</config-property>
<config-property name="DriverClass"type="java.lang.String">
com.microsoft.jdbc.sqlserver.SQLServerDriver
</config-property>
<!--set these only if you want only default logins, not through JAAS-->
<config-property name="UserName" type="java.lang.String">sa</config-property>
<config-property name="Password" type="java.lang.String">sa</config-property>
</properties>
</attribute>
```

设定了数据源的 JndiName 为 MSSQLDS，数据库连接的 URL，数据库的用户名 sa，密码 sa。在客户端输出以下内容：

```
Got context
Total of all accounts in bank initially = 2345.0
Initial Balance = 0.0
After depositing 100, account balance = 100.0
Total of all accounts in bank now = 2445.0
Found account with ID 123456789 Balance = 100.0
Now trying to withdraw $150, which is more than is currently available.  This should
generate an exception..
Caught exception!
account.AccountException: 余额不够
Destroying account..
```

【分析与总结】

本例涉及了 EJB 对数据库的查询、插入、删除等常用操作，如此可见 BMP 需要开发人员控制底层数据操作。

实例 119　MessageDrivenBean

本实例演示如何编写 MessageDrivenBean。

【编程要点】

在 EJB 2.0 中，对规范的一个基础性更改是添加了一种全新的企业级 bean 类型，即 MessageDrivenBean。MessageDrivenBean 专门设计来处理人网的 JMS 消息。JMS 是一种与厂商无关的 API，用来访问消息收发系统。

正像会话 bean 和实体 bean 一样，MessageDrivenBean 也是一种完备的企业级 bean，但其间仍存在一些重要的区别。消息 bean 没有远程接口或本地接口。这是因为消息 bean 不是 RPC 组件。它没有供 EJB 客户机调用的业务方法。消息 bean 监听虚拟消息通道（主题或队列），并使用其他 JMS 客户机发送给该通道的消息。

当部署了一个消息驱动的 bean 以后，它就被指派来处理特定主题或队列中的消息。JMS 客户机（Java 应用程序、bean 或本地客户机）发送的任何消息，将由消息路由器转发给消息 bean，该消息 bean 正是被指派来从该虚拟通道中接收消息的。当一条消息被发送给一个消息 bean 时，EJB 容器就会从某个池中选择该 bean 的一个实例来处理这条消息。当 bean 实例调用其 onMessage()方法时，它就会接收到这条消息，并能够以它认为合适的任何方式来处理这条消息。一旦这条消息被使用，则只要事务没有异常中止，这条消息都不会被传送给这个消息 bean 的任何其他实例。

【程序实现】

MessageBean.java 文件内容如下：

```
//MessageBean.java
//一个接收购物订单的消息驱动 Bean,处理这个订单同时通过 e-mail 的形式
//给客户发一个感谢消息
package examples.Message;
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.*;
import javax.naming.*;
import javax.jms.*;
import java.util.*;
import java.text.*;
import javax.mail.*;
import javax.activation.*;
import javax.mail.internet.*;
//消息驱动 Bean 必须实现两个接口 MessageDrivenBean 和 MessageListener
public class MessageBean implements MessageDrivenBean,
    MessageListener {
    private transient MessageDrivenContext context = null;
    //一个不带参的构造函数
    public MessageBean() {}
    //在对象创建的过程中将被容器调用
    public void setMessageDrivenContext(MessageDrivenContext context)
    {
        this.context = context;
    }
     //onMessage 函数方法接收消息参数，将其强制转型为合适的消息类型
     //同时打印出消息的内容，同时一个 mail note 将被发送给消息发送者
    public void onMessage(javax.jms.Message msg) {
        try {
          if (msg instanceof MapMessage) {
            MapMessage map = (MapMessage) msg;
            System.out.println("Order received: ");
               System.out.println ("Order ID:"+ map.getString("OrderID")+"Item ID:"
                    + map.getInt("ItemID") +   " Quanity: " + map.getInt("Quantity")
                    + " Unit Price: " + map.getDouble("UnitPrice"));
            sendNote (map.getString("emailID"));
          }
          else {
```

```
                System.out.println ("wrong message type");
              }
            } catch (Throwable te) {
              te.printStackTrace();
            }
        }
         //发送一个 e-mail 通知给由 recipient 参数确定的 e-mail 账号
        private void sendNote(String recipient) {
            try {
              //得到名字上下文
              Context initial = new InitialContext();
              //查询 mail 服务器的会话
              javax.mail.Session        session        -        (javax.mail.Session)
    initial.lookup("java:comp/env/MailSession");
              //创建一个新的 mail 对象
              javax.mail.Message msg = new MimeMessage(session);
              //设置 mail 的属性
              msg.setFrom();
                msg.setRecipients(javax.mail.Message.RecipientType.TO,
InternetAddress.parse(recipient, false));
              msg.setSubject("Order Confirmation");
              DateFormat dateFormatter
                 = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.SHORT);
                Date timeStamp = new Date();
              String messageText = "Thank you for your order." + '\n'
                   + "We received your order on " + dateFormatter.format(timeStamp) + ".";
              msg.setText(messageText);
              msg.setSentDate(timeStamp);
              //发送 mail 消息
              Transport.send(msg);
            } catch(Exception e) {
              throw new EJBException(e.getMessage());
            }
        }
        public void ejbCreate() {}
        public void ejbRemove() {}
    }
```

MessageClient.java 文件内容：

```
//消息客户端程序
package examples.Message;
import javax.jms.*;
import javax.naming.*;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class MessageClient {
    public static void main(String[] args) {
        Context              jndiContext = null;
        QueueConnectionFactory  queueConnectionFactory = null;
        QueueConnection        queueConnection = null;
        QueueSession           queueSession = null;
        Queue                  queue = null;
        QueueSender            queueSender = null;
        MapMessage             message = null;
        final int              NUM_MSGS;
        //获取 JNDI 上下文，同时查看连接 factory 和队列
        try {
           jndiContext = new InitialContext();
              Object tmp = jndiContext.lookup("ConnectionFactory");
```

```
                queueConnectionFactory = (QueueConnectionFactory) tmp;
                jndiContext.lookup("java:comp/env/QueueConnectionFactory");
                queue = (Queue) jndiContext.lookup("queue/A");
            } catch (NamingException e) {
                System.out.println("JNDI lookup failed: " + e.toString());
                System.exit(1);
            }
            //通过连接创建会话。创建发送者和映射消息。发送消息，同时对文本进行少量修改
              //发送 end-of-messages 消息，最后关闭连接
            try {
              //创建连接
              queueConnection = queueConnectionFactory.createQueueConnection();
              //通过连接创建会话
              queueSession
=queueConnection.createQueueSession(false,QueueSession.AUTO_ACKNOWLEDGE);
                queueConnection.start();
              //创建发送者
              queueSender = queueSession.createSender(queue);
              //创建 Map 映射消息
              message = queueSession.createMapMessage();
              //定义一些 Name/Value 对
              message.setString ("OrderID", "1");
              message.setInt ("ItemID", 5);
              message.setInt ("Quantity", 50);
              message.setDouble ("UnitPrice", 5.00);
              message.setString ("emailID", "jwu@webjet.com.cn");
              //发送消息
              queueSender.send (message);
            } catch (JMSException e) {
              System.out.println("Exception occurred: " + e.toString());
            }
            finally {
              if (queueConnection != null) {
                try {
                  queueConnection.close();
                } catch (JMSException e) {}
              }
            }
        }
    }
```

【运行结果】

运行服务器端，然后运行客户端。服务器端将显示如下信息：

```
18:40:08,169 INFO [STDOUT] 订单已被接受：
18:40:08,169 INFO [STDOUT] 订单编号：1 条目编号：5 数量：50 单位价格：5.0
```

而且在邮箱里可以发现有新邮件，主题为订单确认。

邮件内容：

```
感谢您的订购！
我们收到您的订单是在 2004 年 11 月 1 日下午 6:40。
```

【分析与总结】

bean 使用的消息不一定要是由其他 bean 所产生的。消息 bean 可以使用由符合 JMS 的厂商提供的任何主题或队列中的消息。消息 bean 使用的消息可以来自其他 bean（会话 bean、实体 bean 或消息 bean）、非 EJB 的 Java 应用程序、或者甚至非 Java 的应用程序（如果其供应商符合 JMS）。例如，遗留应用程序可能使用 IBM 的 MQSeries 向队列发送消息，而该消息既可以由其他遗留应用程序使用，同样可以由消息 bean 使用。

实例 120　JSP 调用 EJB

本实例演示如何在 JSP 中调用 EJB。

【编程要点】

前面已经介绍过 JSP 中使用 JavaBean 的编程方法。而在 JSP 中使用 JavaBean 和用 Enterprise JavaBean 是有区别的。使用 JavaBean 的时候，通常只是使用简单的类来包装数据信息。JavaBean 通过简单的 get 或者 set 方法来存取 JavaBean 的属性值。允许用 jsp:usebean 标签来动态的设定属性值。用 Enterprise JavaBean 的话，过程比较复杂一些，首先要初始化上下文，然后通过名字根据名字服务找到所需要的 EJB，创建一个实例之后，就可以使用了。

一般来讲，JSP 调用 EJB 有两种方式：

（1）JSP+EJB。

在 JSP 的 Java 代码中，直接查找、创建得到 EJB 引用，再调用 EJB 商业方法。

（2）JSP+HomeFactory+EJB。

JSP 调用 HomeFactory 类，通过 HomeFactory 得到 ejbHome 实例，再用 ejbHome 实例创建 EJB 得到引用，再调用 EJB 的商业方法。

这里用到第一种方法，采用的是无状态会话 Bean，沿用本章开头的第二个例子的 EJB。

【程序实现】

BSBrowser.jsp 文件内容如下：

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ page import="javax.naming.Context"%>
<%@ page import="javax.naming.InitialContext"%>
<%@ page import=" myEjb.*"%>
<%@ page import="javax.rmi.PortableRemoteObject"%>
<%@ page import="java.util.Properties"%>
<html>
<head>
<title>
BSBrowser
</title>
</head>
<body>
<h1>
<%="This is BS EJB example!"%><br>
<%
try{
     //为初始化上下文获得一个系统属性对象
  Properties pro=System.getProperties();
  //初始化并得到一个上下文
  Context ctx = new InitialContext(pro);
  //查询 jndi 名，通过强制转型得到 Home 接口
  HelloWorldHome home = (HelloWorldHome) javax.rmi.PortableRemoteObject.narrow
(ctx.lookup ("HelloWorldHome "), HelloWorldHome.class);
  //得到远程接口的引用
  HelloWorld test=home.create();
  out.write(test.printHelloWorld());
}
catch(Exception e)
{
  out.println(e.toString());
}
%>
<%="This is Browser speaking."%><br>
<%="The example is over!"%><br>

</h1>
</body>
</html>
```

【运行结果】

JSP 页面显示结果如下：

```
This is BS EJB example!
Hello World!
This is Browser speaking.
The example is over!
```

【分析与总结】

从本例可以看出调用 EJB 主要用到的是 JNDI 和 RMI 技术。

小结

企业 JavaBeans 规范通过为应用服务器的组件提供标准组织建立 RMI 之上。RMI 服务的 EJB 展开将服务器的任务及功能与服务（或组件）的任务及功能区分开来。EJB 服务器负责管理服务、客户会话和服务组件的安全性、事务处理、创建和消除。本章介绍了几类 EJB 的使用，会话 Bean、实体 Bean 和消息 Bean。读者可以初步了解 EJB 的编程和部署方法。

第 12 章　Java 安全性编程

本章通过 8 个实例介绍了 Java 的安全性编程，其中包括 DES、RSA 加密解密、数字签名、创建数字证书、SSL 通信、HTTPS 服务等。

实例 121　一个简单的加密和解密程序——凯撒密码

本实例实现一个简单的加密解密程序。

【编程要点】

凯撒密码是将字母表中的字母移动一定位置而实现加密。由于它是一种对称密码体制，加解密的密钥是一样的。

下边简单说明一下加解密：

加密过程：

密文：C=M+K　(mod 26)

解密过程：

明文：M=C-K　(mod 26)

【程序实现】

```
package security;
public class Caesar{
    public static void main(String args[]){
        String s = args[0];
        int key = Integer.parseInt(args[1]);     //密钥
        String es="";
        for(int i=0; i<s.length(); i++){
            char c = s.charAt(i);
            if(c>='a' && c<='z'){                //是小写字母
                c += key%26;
                if(c<'a') c += 26;               //向左超界
                if(c>'z') c -= 26;               //向右超界
            }
            else if(c>='A' && c<='Z'){           //是大写字母
                c += key%26;
                if(c<'A') c += 26;
                if(c>'Z') c -= 26;
            }
            es += c;
        }
        System.out.println(es);
    }
}
```

【运行结果】

该程序既可以用于文件加密又可用于解密，只要执行：

```
Java Caesar 明文 密钥
```

即可加密。

在密钥前面加上负号，然后运行：

```
Java Caesar 密文 -密钥
```

即可解密。

如为了加密字符串“Hello World!”，可随意取一个密钥如 4，运行：

```
Java Caesar "Hello World! " 4
```

将输出结果“Lipps Asvph!”。

为了将密文“Lipps Asvph!”解密，需要知道该密文所用的密钥 4，这样，执行：

```
Java Caesar "Lipps Asvph!" -3
```

将得到明文“Hello World!”。

【分析与总结】

凯撒密码是最简单的对称加密解密算法。

主要步骤如下：

（1）读取要加密的字符串、密钥。

（2）取出字符串中每个字符。

（3）对每个字符进行移位。

实例 122 DES 加密和解密

本实例演示如何利用 Java 2 提供的 DES 算法进行加密和解密。

【编程要点】

DES 是一种单钥加密算法，利用单钥加、解密是一个相对简单的过程，在单钥体制下，密钥经过安全的密钥信道由发方传送给收方。单钥密码的特点加密和解密均使用同一密钥，因此其安全性即密钥的安全。

【程序实现】

```
package security;
import java.security.*;
import javax.crypto.*;
public class DESDemo {

    public static void main(String[] args) throws Exception {
        //明文
        byte[] plainText = "Hello World!".getBytes();
        //得到 DES 私钥
        System.out.println("\nStart generating DES key");
        KeyGenerator keyGen = KeyGenerator.getInstance("DES");
        keyGen.init(56);
        Key key = keyGen.generateKey();
        System.out.println("Finish generating DES key");
        //得到 DES cipher 对象，并打印出算法的提供者
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        System.out.println("\n" + cipher.getProvider().getInfo());
        //使用密钥对明文进行加密
        System.out.println("\nStart encryption");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] cipherText = cipher.doFinal(plainText);
        System.out.println("Finish encryption: ");
        System.out.println(new String(cipherText, "UTF8"));
        //使用同一把密钥对密文进行解密
        System.out.println("\nStart decryption");
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] newPlainText = cipher.doFinal(cipherText);
        System.out.println("Finish decryption: ");
        System.out.println(new String(newPlainText, "UTF8"));
    }
}
```

【运行结果】

```
Start generating DES key
Finish generating DES key
SunJCE Provider (implements DES, Triple DES, AES, Blowfish, PBE, Diffie-Hellman,
HMAC-MD5, HMAC-SHA1)
```

```
Start encryption
Finish encryption:
I??I\◄┌?0?%???W?
Start decryption
Finish decryption:
Hello World!
```

【分析与总结】

单钥密码的优点是安全性高且加解密速度快，缺点：一是随着网络规模的扩大，密钥管理成为一个难点；二是无法解决消息确认问题；三是缺乏自动检测密钥泄漏能力。

实例 123　RSA 加密和解密

本程序利用 RSA 算法进行加密与解密。

【编程要点】

RSA 算法是一种非对称加密算法，A 与 B 进行通讯，A 事先生成一对密钥，一个用于加密，称为公钥，一个用于解密，成为密钥。

由于产生这一对密钥的一些数学特性，公钥加密的信息只能用私钥解密。这样，A 只要将公钥对外公开，不论谁就可以使用这个公钥给 A 发送秘密信息了。A 接收到加密信息后可以用私钥打开。

Java 的 KeyPairGenerator 类提供了一些方法来创建密钥对以便用于非对称加密，密钥对创建好后封装在 KeyPair 类型的对象中，在 KeyPair 类中提供获取公钥和私钥的方法。

J2SDK1.4 中只实现了 RSA 密钥的创建，没有实现 RSA 算法，RSA 算法是使用整数进行加密运算的，在 RSA 公钥中包含了两个信息：公钥对应的整数 e 和用于取模的整数 n。对于明文数字 m，计算密文的公式是：$m^e \bmod n$。RSA 算法的解密和加密类似，在 RSA 私钥中包含两个信息：私钥对应的整数 d 和用于取模的整数 n。其中 n 和加密时的 n 完全相同。对于密文数字 c，计算明文的公式是：$c^d \bmod n$，之所以加密时由公式 $m^e \bmod n$ 得到密文 c，通过这个公式计算一下就可以反过来得到原来的明文 m，是由其本身的数学规律决定的。

【程序实现】

```
package security;

import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;
import java.io.*;
import java.math.*;

public class RSADemo {
    public RSADemo() {
    }

    public static void generateKey() {
        try {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(1024);
            KeyPair kp = kpg.genKeyPair();
            PublicKey pbkey = kp.getPublic();
            PrivateKey prkey = kp.getPrivate();
            //保存公钥
            FileOutputStream f1 = new FileOutputStream("pubkey.dat");
            ObjectOutputStream b1 = new ObjectOutputStream(f1);
```

```
            b1.writeObject(pbkey);

            //保存私钥
            FileOutputStream f2 = new FileOutputStream("privatekey.dat");
            ObjectOutputStream b2 = new ObjectOutputStream(f2);
            b2.writeObject(prkey);
        } catch (Exception e) {
        }
    }

    public static void encrypt() throws Exception {
        String s = "Hello World!";
        //获取公钥及参数 e,n
        FileInputStream f = new FileInputStream("pubkey.dat");
        ObjectInputStream b = new ObjectInputStream(f);
        RSAPublicKey pbk = (RSAPublicKey) b.readObject();
        BigInteger e = pbk.getPublicExponent();
        BigInteger n = pbk.getModulus();
        System.out.println("e= " + e);
        System.out.println("n= " + n);
        //获取明文 m
        byte ptext[] = s.getBytes("UTF-8");
        BigInteger m = new BigInteger(ptext);
        //计算密文 c
        BigInteger c = m.modPow(e, n);
        System.out.println("c= " + c);
        //保存密文
        String cs = c.toString();
        BufferedWriter out =
            new BufferedWriter(
                new OutputStreamWriter(new FileOutputStream("encrypt.dat")));
        out.write(cs, 0, cs.length());
        out.close();

    }

    public static void decrypt() throws Exception {
        //读取密文
        BufferedReader in =
            new BufferedReader(
                new InputStreamReader(new FileInputStream("encrypt.dat")));
        String ctext = in.readLine();
        BigInteger c = new BigInteger(ctext);

        //读取私钥
        FileInputStream f = new FileInputStream("privatekey.dat");
        ObjectInputStream b = new ObjectInputStream(f);
        RSAPrivateKey prk = (RSAPrivateKey) b.readObject();
        BigInteger d = prk.getPrivateExponent();

        //获取私钥参数及解密
        BigInteger n = prk.getModulus();
        System.out.println("d= " + d);
        System.out.println("n= " + n);
        BigInteger m = c.modPow(d, n);

        //显示解密结果
        System.out.println("m= " + m);
        byte[] mt = m.toByteArray();
        System.out.println("PlainText is ");
        for (int i = 0; i < mt.length; i++) {
            System.out.print((char) mt[i]);
        }
    }

    public static void main(String args[]) {
        try {
            generateKey();
```

```
                encrypt();
                decrypt();
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }

    }
```

【运行结果】

在当前文件夹中生成 privatekey.dat、publickey.dat、encrypt.dat 三个文件，分别存储私钥、密钥、密文。

控制台运行结果如下：

```
e= 65537
n=
140123134593779081257334227520681949772858771952918228935403447590056899855323514055600
022339647061237728524632144818305406839152704825434779613488040272815539985599591041000
762486092955077149130804639467779548662445571629113854148950346030432838797646576695610
669155170435687111263176367191360468120467061855533
c=
238010723399982177746551940160554866006464584572601164910840223733918798700509895846710
505185419863950888575881278178012742114557797808769809809942087059492637935119058749260
137475510441691277575898713093924715262408268026062058857268508910263886150417838692620
471599304541224664249904808117414058058556019267 95
d=
680528442099179757593373381106212640084276340462553108745434843423397632710543987194360
341463391109165152693976919514448753266611447257085859944414732722488127787494178890150
172038038664618853525765075349764355473900200260901898612075649915826367784524051807730
423410754508483126541925571339695786746787552052 29
n=
140123134593779081257334227520681949772858771952918228935403447590056899855323514055600
022339647061237728524632144818305406839152704825434779613488040272815539985599591041000
762486092955077149130804639467779548662445571629113854148950346030432838797646576695610
669155170435687111263176367191360468120467061855533
m= 22405534230753928650781647905
PlainText is
Hello World!
```

【分析与总结】

网络中的加密普遍采用双钥和单钥密码相结合的混合加密体制，即加解密时采用单钥密码，密钥传递则采用双钥密码，这样解决了密钥管理困难和加解密速度的问题。

实例 124　数字签名

本实例演示数字签名的应用。

【编程要点】

数字签名技术采用了非对称加密（RSA）和报文摘要（Message Digest）算法。发送者使用私钥加密报文摘要(签名),接收者使用公钥解密报文摘要以验证签名是否是某个人的。由于私钥只有加密者才拥有，因此如果接收者用某个公钥解密了某个报文摘要，就可以确定这段报文摘要必然是对应的私钥持有者发来的。

【程序实现】

```
//DigitalSignDemo.java
package security;
import java.io.*;
import java.security.*;
import java.security.interfaces.*;

public class DigitalSignDemo{

    public static void generateKey() {
```

```
        try {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(1024);
            KeyPair kp = kpg.genKeyPair();
            PublicKey pbkey = kp.getPublic();
            PrivateKey prkey = kp.getPrivate();

            //保存公钥
            FileOutputStream f1 = new FileOutputStream("pubkey.dat");
            ObjectOutputStream b1 = new ObjectOutputStream(f1);
            b1.writeObject(pbkey);

            //保存私钥
            FileOutputStream f2 = new FileOutputStream("privatekey.dat");
            ObjectOutputStream b2 = new ObjectOutputStream(f2);
            b2.writeObject(prkey);
        } catch (Exception e) {
        }
    }

    public static void sign() throws Exception{
        //获取要签名的数据
        FileInputStream f = new FileInputStream("msg.dat");
        int num = f.available();
        byte[] data = new byte[num];
        f.read(data);

        //获取私钥
        FileInputStream f2 = new FileInputStream("privatekey.dat");
        ObjectInputStream b = new ObjectInputStream(f2);
        RSAPrivateKey prk = (RSAPrivateKey)b.readObject();
        Signature s = Signature.getInstance("MD5WithRSA");
        s.initSign(prk);
        s.update(data);
        System.out.println("");
        byte[] signeddata = s.sign();

        //打印签名
        for(int i=0; i<data.length; i++){
            System.out.print(signeddata[i] + ",");
        }

        //保存签名
        FileOutputStream f3 = new FileOutputStream("sign.dat");
        f3.write(signeddata);
    }

    public static void checkSign() throws Exception{
        FileInputStream f = new FileInputStream("msg.dat");
        int num = f.available();
        byte[] data = new byte[num];
        f.read(data);

        //读签名
        FileInputStream f2 = new FileInputStream("sign.dat");
        int num2 = f2.available();
        byte[] signeddata = new byte[num2];
        f2.read(signeddata);

        //读公钥
        FileInputStream f3 = new FileInputStream("pubkey.dat");
        ObjectInputStream b = new ObjectInputStream(f3);
        RSAPublicKey pbk = (RSAPublicKey)b.readObject();

        //获取对象
        Signature s = Signature.getInstance("MD5WithRSA");

        //初始化
```

```
            s.initVerify(pbk);

            //传入原始数据
            s.update(data);
            boolean ok = false;
            try{

                //用签名验证原始数据
                ok = s.verify(signeddata);
                System.out.println(ok);
            }
            catch(SignatureException e){
                System.out.println(e);
            }
            System.out.println("Check Over!");
        }

        public static void main(String args[]) throws Exception{
            sign();
            checkSign();
        }
    }
```

【运行结果】

```
60,-121,-94,-43,-55,-85,-107,-40,111,120,126,16,true
Check Over!
```

【分析与总结】

程序中 generateKey()函数生成一对 RSA 密钥分别保存在 pubkey.dat 和 privatekey.dat 中，发送方由 sign()函数将 msg.dat 中的原始数据签名后得到数据保存在 sign.dat 中，接收方收到原始数据 msg.dat 和签名数据 sign.dat 后用公钥检查是否是某人所发。

实例 125　创建数字证书

本实例使用 J2SDK 提供的 keytool 工具创建数字证书。

【编程要点】

数字签名用来确定数据的身份，保证了身份的确定性、不可篡改性和不可否认性。使用数字签名的前提是接收数据者能够确定验证签名时所使用的公钥确实是某个人的，数字证书可以解决这一问题。数字证书的主要功能是保存公钥与某个人或机构的对应关系。

这里利用 keytool 及 RSA 算法和在指定的密钥库 mykeystore 中创建公钥/私钥对和证书。

keytool 的-keyalg 参数可以指定密钥的算法，如果需要指定密钥的长度，可以再加上-keysize 参数。密钥长度默认为 1024 位，使用 DSA 算法时，密钥长度必须在 512 至 1024 之间，并且是 64 的整数倍。

keytool 的-keystore 参数可以指定密钥库的名称。密钥库其实是存放密钥和证书的文件，密钥库对应的文件如果不存在则自动创建。

-validity 参数可以指定所创建的证书有效期是多少天。

使用 keytool 的-export 参数可以将别名指定的证书导出到文件，文件名通过-file 参数指定。

实现：

在命令行中输入 “keytool –genkey –alias mytest –keyalg RSA –keysize 1024 –keystore mykeystore –validity 4000”，将使用 RSA 算法生成 1024 位的公钥/私钥对及整数，密钥长度

为 1024 位，证书有效期为 4000 天。使用的密钥库为 mykeystore 文件。

【运行结果】

运行结果如图 12-1 所示。



图 12-1

由于当前目录下没有 mykeystore 文件，因此以上操作将在当前目录建立文件名为 mykeystore 的文件，并提示输入一个密码加以保护，这里设置为 123456，如图 12-2 所示。



图 12-2

在当前目录下将增加一个文件 mytest.cer，其内容是编码过的，可以在屏幕上显示、复制或打印，如图 12-3 所示。



图 12-3

实例 126　简单的 SSL 通信

本实例演示如何使用 SSL 协议进行通信。

【编程要点】

SSL 编程使用客户机/服务器模式，服务器端程序接受客户程序建立连接，并以加密方式向客户程序发送一字符串 Hello。SSL 服务器程序运行时需要指定密钥库，以便向客户程序证明自己的身份。SSL 客户端程序和服务器端的程序建立连接后，接收其发来的字符串并自动对其进行解密。

SSL 编程和基于 Socket 的编程类似，首先创建 ServerSocket 对象，传入端口号，然后执行 ServerSocket 对象的 accept()方法获取 Socket 类型的对象，并侦听端口以等待客户程序和服务器连接。最后通过 Socket 类型的对象获得输入和输出流，通过输入和输出流与客户程序进行通信。SSL 编程和基于 Socket 的编程不同的地方在于其 ServerSocket 对象是通过一个特殊的对象——SSLServerSocketFactory 类型的对象创建的，这样以后的输入和输出流将自动按照 SSL 协议指定的方法交换密钥并对数据进行加密。此外，需要指定包含证书的密钥库，以便客户程序确定 SSL 服务器是否可靠。

SSL 客户端的编程也和基于 Socket 的客户端编程类似：首先得到 Socket 类型的对象，然后通过 Socket 类型的对象获得输入和输出流，通过输入和输出流与服务器程序进行通信。和服务器程序类似，SSL 客户端编程和基于 Socket 的客户端编程不同的地方在于其 Socket 对象是通过一个特殊的对象 SSLSocketFactory 类型的对象创建的。

【程序实现】

服务器端程序 SSLServer.java 内容如下：

```
package security;
import java.net.*;
import javax.net.ssl.*;
import java.io.*;

public class SSLServer{
    public static void main(String args[]) throws Exception{
        System.setProperty("javax.net.ssl.keyStore","mykeystore");
        System.setProperty("javax.net.ssl.keyStorePassword", "123456");
        SSLServerSocketFactory ssf = (SSLServerSocketFactory)
            SSLServerSocketFactory.getDefault();
        ServerSocket ss = ssf.createServerSocket(5432);
        System.out.println("Waiting for connection...");
        while(true){
            Socket s = ss.accept();
            PrintStream out = new PrintStream(s.getOutputStream());
            out.println("Hello");
            out.close();
            s.close();
        }
    }
}
```

客户机端程序 SSLClient.java 内容如下：

```
package security;

import java.io.*;
import javax.net.ssl.*;
import java.net.*;

public class SSLClient{
    public static void main(String args[]) throws Exception{
        System.setProperty("javax.net.ssl.trustStore", "clienttrust");
        SSLSocketFactory ssf = (SSLSocketFactory)SSLSocketFactory.getDefault();
        Socket s = ssf.createSocket("127.0.0.1", 5432);
        BufferedReader in = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
        String x = in.readLine();
        System.out.println(x);
        in.close();
```

```
        }
    }
```

【运行结果】

由于 SSL 协议需要通过数字证书向客户表明服务器是否值得信任，因此当前目录下必须有密钥库，本实例将上例产生的密钥库 mykeystore，将其复制到当前目录下，然后运行程序，控制台输出："Waiting for connection..."，此时开始等待客户程序的连接。

为了使客户机程序能够顺利验证该服务器提供的证书，应该把 mykeystore 中所使用的证书或其签发者的证书 mytest.cer 提供给客户程序。将该文件存放在客户程序所在的目录中，以便客户程序向服务器程序确认信任该证书。为了在程序中使用，须将该证书导入密钥库，操作如图 12-4 所示。

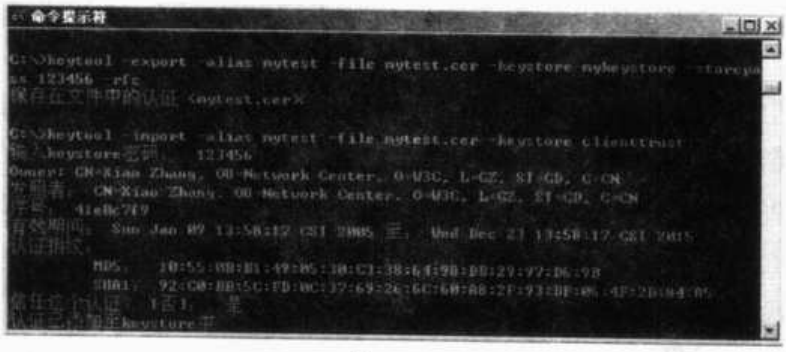


图 12-4

该操作将证书 mytest.cer 导入密钥库 clienttrust 中。

运行服务器端程序，控制台输出：

```
Waiting for connection…
```

运行客户机端程序，控制台输出：

```
Hello
```

【分析与总结】

SSL 协议已广泛用于 Web 浏览器与服务器间的身份认证和加密数据传输，该协议位于 TCP/IP 协议与各应用层协议间，为数据通信提供安全支持。

实例 127　HTTPS 服务

本实例介绍使用 SSL 加密 HTTP 流量的 HTTPS。

【编程要点】

HTTPS 使用的标准端口号是 443，因此服务器程序应该使用该端口号，这样浏览器就可以输入"https://服务器地址"来访问服务器了。如果服务器程序中的端口号不是使用 443，则浏览器访问时应通过"https://服务器地址：端口号"来访问服务器。获得 Socket 类型对象后，就可以和以前一样得到输入/输出流，不同的只是输入/输出流的处理方式不一样，要按照 HTTP 协议规定的方式进行通信。

【程序实现】

```
//HttpsServer.java
package security;
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;
```

```
import java.util.*;
public class HttpsServer
{
    //keystore文件名
    String keystore = "mykeystore";
    //keystore密码
    char keystorepass[] = "123456".toCharArray();
    //key密码
    char keypassword[] = "123456".toCharArray();
    //使用缺省的433端口
    public static final int HTTPS_PORT = 443;

    public ServerSocket getServer() throws Exception
    {
        KeyStore ks = KeyStore.getInstance("JKS");
        ks.load(new FileInputStream(keystore), keystorepass);
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
        kmf.init(ks, keypassword);
        SSLContext sslcontext = SSLContext.getInstance("SSLv3");
        sslcontext.init(kmf.getKeyManagers(), null, null);
        ServerSocketFactory ssf = sslcontext.getServerSocketFactory();
        //建立SSL Socket连接
        SSLServerSocket serversocket =
            (SSLServerSocket) ssf.createServerSocket(HTTPS_PORT);
        return serversocket;
    }
    public void run()
    {
        ServerSocket listen;
        try
        {
            //为每一个连接建立线程
            listen = getServer();
            while (true)
            {
                Socket client = listen.accept();
                ProcessConnection cc = new ProcessConnection(client);
            }
        }
        catch (Exception e)
        {
            System.out.println("Exception: " + e.getMessage());
        }
    }
    public static void main(String argv[]) throws Exception
    {
        HttpsServer https = new HttpsServer();
        https.run();
    }
}
class ProcessConnection extends Thread
{
    Socket client;
    BufferedReader is;
    DataOutputStream os;
    public ProcessConnection(Socket s)
    {
        client = s;
        try
        {
            //客户端请求流
            is =
                new BufferedReader(
                    new InputStreamReader(client.getInputStream()));
            //服务器响应流
            os = new DataOutputStream(client.getOutputStream());
        }
        catch (IOException e)
```

```
            {
                System.out.println("Exception: " + e.getMessage());
            }
            this.start();
        }
        public void run()
        {
            try
            {
                //得到客户端请求并解析请求
                String request = is.readLine();
                System.out.println("Request: " + request);
                StringTokenizer st = new StringTokenizer(request);
                if ((st.countTokens() >= 2) && st.nextToken().equals("GET"))
                {
                    if ((request = st.nextToken()).startsWith("/"))
                        request = request.substring(1);
                    if (request.equals(""))
                        request = request + "index.html";
                    File f = new File(request);
                    shipDocument(os, f);
                }
                else
                {
                    os.writeBytes("400 Bad Request");
                }
                client.close();
            }
            catch (Exception e)
            {
                System.out.println("Exception: " + e.getMessage());
            }
        }
        public static void shipDocument(DataOutputStream out, File f)
            throws Exception
        {
            try
            {
                DataInputStream in = new DataInputStream (new FileInputStream(f));
                int len = (int) f.length();
                byte[] buf = new byte[len];
                in.readFully(buf);
                in.close();
                out.writeBytes("HTTP/1.0 200 OK\r\n");
                out.writeBytes("Content-Length: " + f.length() + "\r\n");
                out.writeBytes("Content-Type:text/html\r\n\r\n");
                out.write(buf);
                out.flush();
            }
            catch (Exception e)
            {
                //回写错误信息
                out.writeBytes("<html><head><title>error</title>"+
                        "</head><body>\r\n\r\n");
                out.writeBytes("HTTP/1.0 400 " + e.getMessage() + "\r\n");
                out.writeBytes("Content-Type: text/html\r\n\r\n");
                out.writeBytes("</body></html>");
                out.flush();
            }
            finally
            {
                out.close();
            }
        }
    }
```

【运行结果】

本实例仍然采用 mykeystore 文件，密码均为 123456。运行程序，在浏览器中输入 https://localhost/index.html，如果无响应，则加上端口号 433，显示的安全警报如图 12-5 所

示。

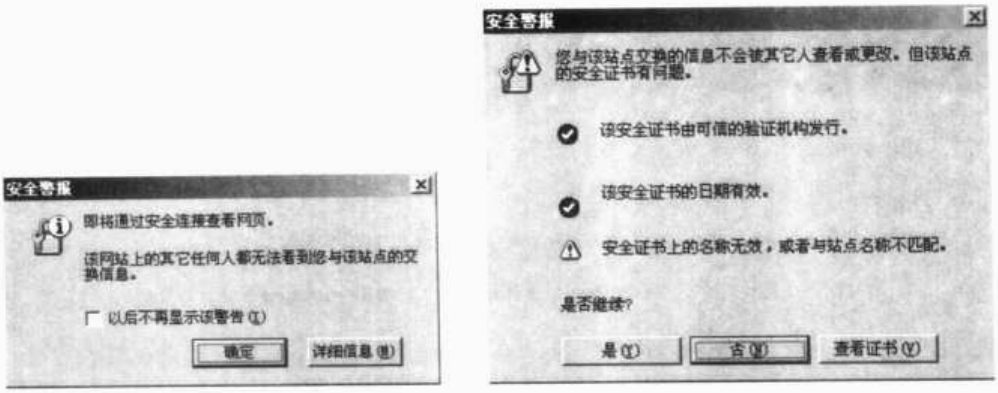


图 12-5

这是因为 HTTP 服务器证书由未知的 CA 创建，在浏览器保存的 CA 中未找到这个 CA。单击“是”按钮，浏览器显示相应目录下的 index.html 文件内容。控制台输出：

```
Request: null
Exception: null
Request: GET /index.html HTTP/1.1
```

【分析与总结】

HTTP 协议基本的规则是：服务器先读取浏览器发来的数据（类似“GET /test.html HTTP/1.1”的字符串），如果是 GET 请求，则根据请求的内容，向浏览器发送 HTTP 版本、MIME 版本、数据类型、数据长度、一个空行以及数据内容等信息。浏览器只有接收到这样的信息，才认为 HTTP 协议执行正确，从而将后面发送的内容作为网页内容显示出来。

实例 128　密码输入框

本实例演示了图形界面中如何控制口令的回显。

【编程要点】

Swing 提供了 JPasswordField 类用于输入一行口令，通过其 setEchoChar()方法可以设置回显的字符，通过其 getPassword()方法可以获得输入的口令值。

如果输入一般的文本，不需要设置回显的字符，则可以使用 JTextField 类。通过其 getText()方法获得用户输入的文本。

【程序实现】

```
//PasswordDemo.java
package security;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PasswordDemo{
    //口令和文本输入域
    JPasswordField passwordField = new JPasswordField(12);
    JTextField textField = new JTextField(12);

    public static void main(String[] args){
        PasswordDemo pd = new PasswordDemo();
        pd.go();
    }

    void go(){
        //设置口令回显字符为 x
```

```
        passwordField.setEchoChar('X');
        //创建各个标签、按钮
        final JFrame f = new JFrame("PasswordDemo");
        JLabel label1 = new JLabel("输入用户名: ");
        JLabel label2 = new JLabel("输入口令");
        JButton button1 = new JButton("确定");
        JButton button2 = new JButton("取消");

        //鼠标单击 "OK" 按钮则执行这段代码
        button1.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                //获取输入的名字
                String name = textField.getText();
                //获取输入的口令值
                char[] password = passwordField.getPassword();
                System.out.println("输入用户名: " + name);
                System.out.println("输入口令: "+ new String(password));
            }
        });

        //使用网络布局
        JPanel contentPane = new JPanel(new GridLayout(3,2));
        contentPane.add(label1);
        contentPane.add(textField);
        contentPane.add(label2);
        contentPane.add(passwordField);
        contentPane.add(button1);
        contentPane.add(button2);
        f.setContentPane(contentPane);

        //关闭窗口
        f.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
        f.pack();
        f.setVisible(true);
    }
}
```

【运行结果】

运行结果如图 12-6 所示。

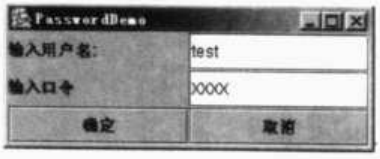


图 12-6

单击“确定”按钮后，控制台输出：

```
输入用户名: test
输入口令: test
```

【分析与总结】

这个实例在实际应用中经常可以用到。

小结

作为一种在 Internet 上广泛应用的编程语言，Java 语言的安全性在其客户端和服务器

端编程中显得至关重要。Java 不仅从其自身语言特性上保证其安全，如取消了指针的使用，数组越界的自动检查，只允许合法的类型转换，而且还从 Java 虚拟机、API 类库的设计上保证 Java 程序在 Internet 上能安全稳定地运行。

本章介绍了 Java 的几种加密解密算法的调用，数字签名的应用，而且还介绍了基于 SSL 和 HTTPS 的编程。通过本章学习，读者可以解决数据安全传输，服务器和用户身份的确定等问题。

第 13 章　J2ME 编程

本章通过 7 个实例介绍了 Java 2 Platform Micro Edition 编程，其中包括 Command 对象应用、查看内存利用情况、菜单设计、记录存储、使用定时器、无线 HTTP 连接等。

实例 129　Hello World!

本实例演示如何编写和运行第一个 J2ME 程序。

【编程要点】

Sun Microsystems 将 J2ME 定义为“一种以广泛的消费性产品为目标的高度优化的 Java 运行时环境，包括寻呼机、移动电话、可视电话、数字机顶盒和汽车导航系统”。自从 1999 年 6 月在 JavaOne Developer Conference 上声明之后，J2ME 为小型设备带来了 Java 语言的跨平台功能，允许移动无线设备共享应用程序。有了 J2ME，Sun 已经使 Java 平台能够适应集成了或基于小型计算设备的用户产品。

J2ME 使用配置和简表定制 Java 运行时环境（JRE）。作为一个完整的 JRE，J2ME 由配置和简表组成，配置决定了使用的 JVM，而简表通过添加特定于域的类来定义应用程序。

配置将基本运行时环境定义为一组核心类和一个运行在特定类型设备上的特定 JVM。Sun 提供的 J2ME 配置是适合不同层次的市场需求的——CLDC 适合小型设备，而 CDC 适合大型设备。J2ME 环境可以被动态地配置为提供运行应用程序所需要的环境，而不用考虑是否为设备提供了运行该应用程序所需的所有 Java 技术库。核心平台接收应用程序代码和库。运行在网络上的服务器软件执行配置工作。

简表定义应用程序，特别地，它向 J2ME 配置中添加特定于域的类，定义设备的某种作用。一个简表定义了受支持设备的类型。例如，移动信息设备简表（MIDP），定义了蜂窝电话的类。它把一些特定于域的类加入 J2ME 配置中来定义对类似设备的使用。它已经为 J2ME 定义了两个简表：KJava 和 MIDP。它们都构建在 CLDC 之上。KJava 和 MIDP 都和 CLDC 及小型设备相关联。简表被构建在配置的顶部。由于简表是特定于运行应用程序的设备的大小（内存的数量），所以，某个简表是与某种特定的配置相关联的。

图 13-1 描述了不同的虚拟机、配置和简表之间的关系。它同时把 J2SE API 和它的 Java 虚拟机进行了比较。虽然 J2SE 虚拟机通常被称为一种 JVM，但是 J2ME 虚拟机、KVM 和 CVM 都是 JVM 的子集。KVM 和 CVM 均可被看作是一种 Java 虚拟机——它们是 J2SE JVM 的压缩版，并特定于 J2ME。

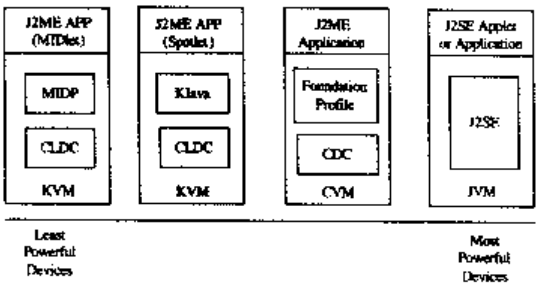


图 13-1

J2ME Wireless Toolkit（WTK）是常用的 J2ME SDK，使用 WTK 可以进行程序的编译、校验、运行。目前最新版本是 WTK2.1，该版本支持 JTWI（无线产业 Java 技术规范）Roadmap 1、CLDC1.1 和 J2ME Web Services API。WTK2.1 需要 JDK1.4 版本的支持，先从 Sun 公司主页下载 JDK 安装后，再安装 WTK2.1，安装 WTK 时会询问 JDK 的安装路径，请在这一步指定正确的 JDK 路径。

javax.microedition.lcdui——定义用来控制 UI 的类。这个软件包既包含高级 UI 类（例如 Form、Command、DateField 和 TextField 等），又包含低级 UI 类（允许用低级方式控制 UI）。

javax.microedition.midlet——包含 MIDP 主类中的一个 MIDlet 类，为 MIDP 应用程序提供访问关于其运行所在环境信息的权限。MIDlet 是一个 Java 类，它扩展了 javax.microedition.midlet.MIDlet 抽象类。实现 startApp()、pauseApp()和 destroyApp()方法，这些方法类似于 J2SE 的 java.applet.Applet 类中的 start()、stop()和 destroy()方法。

除了扩充 javax.microedition.midlet.MIDlet 的主 MIDlet 类之外，一个 MIDP 应用程序通常还包括其他一些类，这些类能随它们的资源一起被打包成为 jar 文件——称之为 MIDlet 套件。

一个 MIDlet 套件中的不同 MIDlet 能共享 jar 文件的资源，尽管不同套件中的 MIDlets 不能直接相互作用。

MIDlet 在应用程序生命周期中有三种可能的存在状态——运行状态、暂停状态、销毁状态。运行状态，正如其名称所暗示的，意味着 MIDlet 正在运行中。这种状态始于 startApp 方法被调用时。在暂停状态中，MIDlet 持有的所有资源将被释放，但是它准备着再次被运行。调用 notifyPaused 方法时，MIDlet 处于暂停状态。在销毁状态中，MIDlet 已经永久地将其自身关闭，释放所有的资源，等待着废物清理程序的处理。它是通过 notifyDestroyed 方法来调用的。

【程序实现】

```
//HelloWorld.java
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class HelloWorld extends MIDlet
{
    private Display display;
    public HelloWorld() {
    }

    public void startApp() {
        display = Display.getDisplay(this);
        TextBox t = new TextBox("Demo App", "Hello World", 256, 0);
      //将 TextBox 作为当前屏幕的显示对象
        display.setCurrent(t);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

【运行结果】

在 ktoolbar 界面中，单击“New Project”按钮，在弹出的对话框中输入如图 13-2 所示

的数据。

然后单击“Create Project”按钮，将出现如图 13-3 所示的工程设置对话框。在该对话框中可以设置工程的各种属性，此时先接受默认设置，单击“OK”按钮。

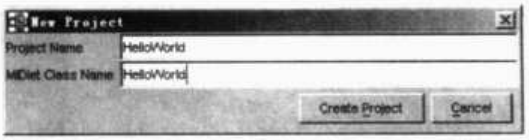


图 13-2

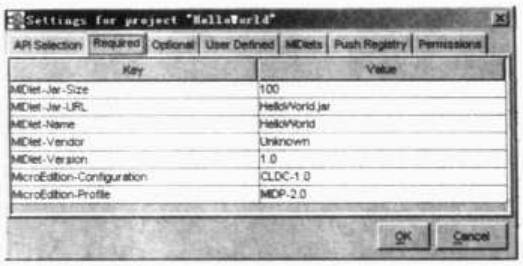


图 13-3

WTK 把所有工程都放在 WTK21\apps\目录下。创建工程后检查该目录，会发现多出一个 HelloWorld 文件夹。在该文件夹中又有四个文件夹：src、res、lib 和 bin。其中“src”用于存放源文件，“res”存放程序所需的资源文件，“lib”存放程序所需的库文件，刚创建工程时这三个目录下没有任何文件。在“bin”目录下有两个文件，“MANIFEST.MF”和“HelloWorld.jad”。“MANIFEST.MF”是清单文件，记录了工程的各种属性信息，“HelloWorld.jad”记录的内容与“MANIFEST.MF”类似，使用手机从网上下载 Java 软件时会用到该文件。这两个文件的内容都可以用普通的编辑程序修改，也可以在图 13-3 所示的工程设置对话框中修改。在创建工程后，选择“Project”→“Settings…”菜单命令或单击工具栏上的“Settings”按钮，都可以打开工程设置窗口。

将 Hello.java 文件复制到 HelloWorld\src 目录下，然后单击 ktoolbar 上的“Build”按钮编译源程序。如果源程序有错误，控制台会显示相应的错误信息。如果编译成功会在控制台显示如下信息：

```
Creating project "HelloWorld"
Place Java source files in "C:\WTK21\apps\HelloWorld\src"
Place application resource files in "C:\WTK21\apps\HelloWorld\res"
Place application library files in "C:\WTK21\apps\HelloWorld\lib"
Settings updated
Project settings saved
Project settings saved
Building "HelloWorld"
Build complete
```

源程序编译成功后，单击工具栏上的“Run”按钮运行程序，单击模拟器上的“Select”按钮进入程序，如图 13-4 所示。



图 13-4

【分析与总结】

此例将带领读者进入精彩的J2ME编程世界。

实例130 Command对象

本实例演示如何在MIDP中使用Command对象。

【编程要点】

J2ME的事件系统比较特殊，必须首先定义一系列的命令（例如Form、Alert、List、TextBox），然后注册到容器对象中再设定命令监听者，编写好commandAction()方法即可。当系统发送某个命令，便由commandAction()方法进行统筹处理。

【程序实现】

```
//CMD.java

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class CMD extends MIDlet implements CommandListener {
    private Display display;
    private Form props;
    //MIDP总共八个命令都包含在此
    private Command backCommand = new Command("BACK", Command.BACK, 2);
    private Command cancelCommand = new Command("CANCEL", Command.CANCEL, 1);
    private Command exitCommand = new Command("EXIT", Command.EXIT, 1);
    private Command helpCommand = new Command("HELP", Command.HELP, 1);
    private Command itemCommand = new Command("ITEM", Command.ITEM, 1);
    private Command okCommand = new Command("OK", Command.OK, 1);
    private Command screenCommand = new Command("SCREEN", Command.SCREEN, 1);
    private Command stopCommand = new Command("STOP", Command.STOP, 1);

    public CMD() {
        display = Display.getDisplay(this);
    }

    public void startApp() {
        props = new Form("Hello World");
        props.append("Hello World!\n");
        props.addCommand(backCommand);
        props.addCommand(cancelCommand);
        props.addCommand(exitCommand);
        props.addCommand(helpCommand);
        props.addCommand(itemCommand);
        props.addCommand(okCommand);
        props.addCommand(screenCommand);
        props.addCommand(stopCommand);
        props.setCommandListener(this);
        display.setCurrent(props);
    }

    public void showScreen(String cmd) {
        Form form = new Form("show cmd");
        form.append(cmd);
        form.addCommand(exitCommand);
        form.setCommandListener(this);
        display.setCurrent(form);
    }

    //对不同的命令进行事件处理
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        } else if (c == helpCommand) {
            showScreen("help");
```

```
            } else if (c == backCommand) {
                showScreen("back");
            } else if (c == cancelCommand) {
                showScreen("cancel");
            } else if (c == itemCommand) {
                showScreen("item");
            } else if (c == okCommand) {
                showScreen("ok");
            } else if (c == screenCommand) {
                showScreen("screen");
            }
            if (c == stopCommand) {
                showScreen("stop");
            }
        }

        public void destroyApp(boolean unconditional) {
        }
        public void pauseApp() {
            display.setCurrent(null);
            props = null;
        }

    }
```

【运行结果】

CMD.java 的运行效果如图 13-5 所示（当屏幕出现 Hello World 字样的时候，需要按下退出键，命令菜单就会出现，可以依次执行各个命令）。



图 13-5

【分析与总结】

程序中定义了八个命令。如果 commandAction()方法接到这八个命令，多半是调用 showScreen()方法，将这几个命令输出。showScreen()方法会产生一个新的容器对象（form），作为当前的屏幕，并把截获的命令显示在屏幕中。

实例 131 查看内存利用

本实例实现 MIDP 中查询内存利用情况的功能。

【编程要点】

通过 Runtime 来查看 Midlet Suite 的容量，Runtime 中相关的两个方法如下：

public native long totalMemory()，返回 Java Virtual Machine 的内存总数。返回以字节

计算的供分配对象使用的当前可用内存总数。

public native long freeMemory()，返回系统的空闲内存数。该方法的返回值总是小于 totalMemory 方法的返回值。

【程序实现】

```
//Memory.java
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Memory extends MIDlet implements CommandListener {
    private Display display;
    private Form props;
    private Command exitCommand = new Command("Exit", Command.EXIT, 1);
    public Memory() {
        display = Display.getDisplay(this);
    }

    public void startApp() {
        props = new Form("Runtime Information");
        //总内存空间大小
        long total = Runtime.getRuntime().totalMemory();
        //剩余内存空间大小
        long free = Runtime.getRuntime().freeMemory();
        props.append("total memory:" + total + "\n");
        props.append("free memory:" + free + "\n");
        props.append("used memory:" + (total-free) +"\n");

        props.addCommand(exitCommand);
        props.setCommandListener(this);
        display.setCurrent(props);
    }

    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }

    public void destroyApp(boolean unconditional) {
    }

    public void pauseApp() {
        display.setCurrent(null);
        props = null;
    }
}
```

【运行结果】

运行结果如图 13-6 所示。



图 13-6

【分析与总结】

这个程序主要是利用 Runtime 类的 totalMemory()方法以及 freeMemory()方法。J2ME 中的 Runtime 类已经不再具有执行外部程序的功能。

实例 132 菜单设计

本实例设计手机上一些常用的菜单。

【编程要点】

List 属于 javax.microedition.lcdui 包，它和 Form 一样，同样属于容器类型的对象。属于容器类型的对象还有 TextBox 和 Alert。

List 对象实际上代表一个选择列表。

List 类的构造函数 List(String arg0,int arg1,String[] arg2, Image[] arg3)：arg0 是选择列表的名字，arg1 是选择列表的形式，Choice.EXCLUSIVE 表示这个选择列表只能够单选。如果是 Choice.MULTIPLE，则表示这个选择列表可以多选，arg2 表示列表包含的表项，arg3 表示各个表项对应的图标。

List 类的 append()方法有两个参数，第一个参数是选择项的描述，第二个参数是一个 Image 对象，代表每个选择项前面的小图标。第二个参数可以是 null 值，但是第一个参数是必须的。

同样可以使用 addCommand()方法往 List 中注册命令，也可以使用 setCommandListener()方法指定命令监听者，这和 Form 是一样的。

在 startApp()方法的最后，使用 Display 对象的 setCurrent()方法将 List 对象设定为当前的屏幕显示对象。

Ticker 对象是一个项目类型的对象，它的作用相当于一个滚动消息栏，在屏幕的上方显示滚动的信息。

Ticker 类的构造函数仅有一个参数，那就是需要滚动显示的消息。

【程序实现】

```
//MIDPApplication.java
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class MIDPApplication extends MIDlet implements CommandListener {
    Ticker newsTicker;              //设置头条新闻用
    private List menuList;          //主菜单
    private List luckyList;         //一周运势菜单
    private List commList;          //通讯簿菜单
    private Command exitCommand;    //离开
    private Command execCommand;    //执行
    private Command menuCommand;    //主目录
    private Command commCommand;    //通讯簿
    private Command luckyCommand;   //一周运势
    private Command saveCommand;    //存储
    String[] menuItem = { "通讯簿", "一周运势" };
    String[] commItem = { "新增", "查询", "修改", "删除" };
    String[] luckyItem = { "金牛座", "处女座", "摩羯座", "射手座", "牡羊座", "狮子座", "
双鱼座",
```

```
            "巨蟹座", "天蝎座", "双子座", "天秤座", "水瓶座" };
        Image[] menuImage = { createImage("/Sokoban.png"),
                createImage("/Tiles.png") };
        private Display display;
        private String currentScreen = "";

        public MIDPApplication() {
            display = Display.getDisplay(this);
            newsTicker = new Ticker("动感地带国庆充值大奉送，充300送300，热烈庆祝北京申奥成
功！");
            exitCommand = new Command("离开", Command.EXIT, 1);
            execCommand = new Command("执行", Command.OK, 1);
            menuCommand = new Command("主目录", Command.SCREEN, 1);
            commCommand = new Command("通讯簿", Command.SCREEN, 1);
            luckyCommand = new Command("运势", Command.SCREEN, 1);

        }

        public void startApp() {
            menuList = new List(null, List.IMPLICIT, menuItem, menuImage);
            menuList.setTicker(newsTicker);
            menuList.addCommand(exitCommand);
            menuList.setCommandListener(this);
            display.setCurrent(menuList);
            currentScreen = "主目录";

        }

        public void pauseApp() {
        }

        public void destroyApp(boolean unconditional) {
        }

        //"命令"事件处理
        public void commandAction(Command c, Displayable s) {
            if (c == exitCommand) {
                destroyApp(false);
                notifyDestroyed();
            }

            if (c == List.SELECT_COMMAND) {
                if (currentScreen == "主目录") {
                    int itemIndex = menuList.getSelectedIndex();
                    switch (itemIndex) {
                    case 0: {
                        doCommunication();
                        break;
                    }
                    case 1:
                        doLucky();
                    }
                } else if (currentScreen == "通讯簿") {
                    int itemIndex = commList.getSelectedIndex();
                    switch (itemIndex) {
                    case 0: {
                        doAppend();
                        break;
                    }
                    case 1: {
                        doQuery();
                        break;
                    }
                    case 2: {
                        doModification();
                        break;
                    }
```

```
                case 3: {
                    doDelete();
                    break;
                }
                }

            } else {

            }
        }

        if (c == menuCommand) {
            display.setCurrent(menuList);
            currentScreen = "主目录";
        }

        if (c == commCommand) {
            doCommunication();

        }

        if (c == luckyCommand) {
            doLucky();
        }

        if (c == menuCommand) {
            doMenu();
        }

        if (c == execCommand) {
            if (currentScreen == "一周运势") {
                showLucky();
            }
        }
    }
    private Image createImage(String name) {
        Image aImage = null;
        try {
            aImage = Image.createImage(name);
        } catch (IOException e) {
        }
        return aImage;
    }

    //显示主目录
    private void doMenu() {
        currentScreen = "主目录";
        display.setCurrent(menuList);
    }

    //显示通讯录目录
    private void doCommunication() {
        Image[] commIcon = { createImage("/StarCruiser.png"),
                createImage("/Stock.png"), createImage("/ManyBalls.png"),
                createImage("/Tiles.png") };

        commList = new List("通讯簿", List.IMPLICIT, commItem, commIcon);

        commList.addCommand(luckyCommand);
        commList.addCommand(menuCommand);
        commList.setCommandListener(this);
        currentScreen = "通讯簿";
        display.setCurrent(commList);
    }

    //显示所有星座
    private void doLucky() {
        luckyList = new List("一周运势", List.EXCLUSIVE,
```

```
                                luckyItem, null);
        luckyList.addCommand(commCommand);
        luckyList.addCommand(menuCommand);
        luckyList.addCommand(execCommand);
        luckyList.setCommandListener(this);
        currentScreen = "一周运势";
        display.setCurrent(luckyList);
    }

    //显示对应的星座的运势
    private void showLucky() {
        int selectedItem = luckyList.getSelectedIndex();
        Alert alert;
        String information = "";
        switch (selectedItem) {
        case 0: {
            information = "金牛座";
            break;
        }
        case 1: {
            information = "处女座";
            break;
        }
        case 2: {
            information = "摩羯座";
            break;
        }
        case 3: {
            information = "射手座";
            break;
        }
        case 4: {
            information = "牡羊座";
            break;
        }
        case 5: {
            information = "狮子座";
            break;
        }
        case 6: {
            information = "双鱼座";
            break;
        }
        case 7: {
            information = "巨蟹座";
            break;
        }
        case 8: {
            information = "天蝎座";
            break;
        }
        case 9: {
            information = "双子座";
            break;
        }
        case 10: {
            information = "天秤座";
            break;
        }
        case 11: {
            information = "水瓶座";
            break;
        }
        }

        alert = new Alert("信息", information, null, AlertType.INFO);
```

```
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
    }

    //处理通讯录添加
    private void doAppend() {
        Form appendForm = new Form("新增");
        saveCommand = new Command("存储", Command.SCREEN, 1);
        TextField nameField = new TextField("姓名", null, 10, TextField.ANY);
        TextField EMailField = new TextField("E Mail", null, 10,
                TextField.EMAILADDR);
        TextField ageField = new TextField("年龄", null, 10, TextField.NUMERIC);
        appendForm.append(nameField);
        appendForm.append(EMailField);
        appendForm.append(ageField);
        appendForm.addCommand(saveCommand);
        appendForm.addCommand(commCommand);
        appendForm.addCommand(luckyCommand);
        appendForm.setCommandListener(this);
        currentScreen = "通讯簿新增";
        display.setCurrent(appendForm);
    }

    //处理通讯录修改
    private void doModification() {
        Form modificationForm = new Form("修改");
        TextField nameField = new TextField("姓名", null, 10, TextField.ANY);
        modificationForm.append(nameField);
        modificationForm.addCommand(execCommand);
        modificationForm.addCommand(commCommand);
        modificationForm.addCommand(luckyCommand);
        modificationForm.setCommandListener(this);
        currentScreen = "通讯簿修改";
        display.setCurrent(modificationForm);
    }

    //处理通讯录查询
    private void doQuery() {
        Form queryForm = new Form("查询");
        TextField nameField = new TextField("姓名", null, 10, TextField.ANY);
        queryForm.append(nameField);
        queryForm.addCommand(execCommand);
        queryForm.addCommand(commCommand);
        queryForm.addCommand(luckyCommand);
        queryForm.setCommandListener(this);
        currentScreen = "通讯簿查询";
        display.setCurrent(queryForm);
    }

    //处理通讯录删除
    private void doDelete() {
        Form deleteForm = new Form("删除");
        TextField nameField = new TextField("姓名", null, 10, TextField.ANY);
        deleteForm.append(nameField);
        deleteForm.addCommand(execCommand);
        deleteForm.addCommand(commCommand);
        deleteForm.addCommand(luckyCommand);
        deleteForm.setCommandListener(this);
        currentScreen = "通讯簿删除";
        display.setCurrent(deleteForm);
    }
}
```

【运行结果】

运行结果如图 13-7 所示。



图 13-7

【分析与总结】

注意本例的通信簿增加、删除、修改等操作都只是作为一个界面，并未实现其功能的。这里实际上主要是 List 和 Command 对象的应用，理解了这两个类就不难做出想要的菜单来。

实例 133　记录存储

本实例演示如何在 J2ME 记录存储中操作。

【编程要点】

J2ME 记录管理系统（RMS）提供了一种机制，通过这种机制 MIDlet 能够持久存储数据，并在以后检索数据。在面向记录的方法中，J2ME RMS 由多个记录存储构成。可以将每个记录存储想像成一个记录集合，它将跨多个 MIDlet 调用持久存在。设备平台负责在平台正常使用的整个过程（包括重新启动、换电池等）中，尽全力维护 MIDlet 的记录存储的完整性。

记录存储在与平台相关的位置（比如非易失性设备存储器）创建，这些位置不直接公开给 MIDlet。RMS 类调用特定于平台的本机代码，这种本机代码使用标准 OS 数据管理器函数来执行实际的数据库操作。

记录存储实现确保所有单个的记录存储操作都是原子的、同步的以及序列化的，因此多个访问将不会毁坏数据。记录存储被盖上时间戳来指示它上次被修改的时间。记录存储还维护版本（version）的完整性，记录值是一个整数，修改记录存储内容的操作每发生一次，这个数加一。版本和时间戳对于同步目的很有用。

当 MIDlet 使用多个线程访问一个记录存储时，协调该访问是 MIDlet 的责任；如果它不能这样做，可能出现无法意料的结果。同样，如果一个平台使用试图同时访问记录存储的多个线程执行记录存储的同步，那么对 MIDlet 及其同步引擎之间的记录存储实施排外访问是平台的责任。

记录存储中的每个记录是一个字节数组，并且有惟一的整数标识符。

javax.microedition.rms.RecordStore 类代表 RMS 记录存储。它提供了几个方法来管理以及插入、更新和删除记录存储中的记录。

要打开一个记录存储，调用 javax.microedition.rms.RecordStore 的 openRecordStore()方

法。public static RecordStore openRecordStore（String recordStoreName, boolean createIfNecessary）打开具有指定名称 recordStoreName 的记录存储。如果没有具有这个名称的记录存储，那么调用这个方法来创建一个。如果记录存储已经打开，这个方法将返回对同一个记录存储对象的引用。

一旦所有操作完成，对 closeRecordStore()的调用将关闭指定名称的记录存储。当一个记录存储被关闭时，不能进行进一步的操作。

通过调用 deleteRecordStore()方法可以删除指定名称的记录存储。

MIDlet 调用 javax.microedition.rms.RecordStore 类的 addRecord()方法来将一条新记录插入到记录存储中。这是阻塞的原子操作，并返回新记录的 recordId。在这个方法返回之前，记录被写到持久存储中。public int addRecord(byte[] data, int offset, int numBytes)插入一条由字节数组 data 代表的记录，这个数组以 offset 作为它的起始索引，numBytes 作为它的长度。

更新一条特殊记录包括获取这个记录的句柄以及设置新信息。public int getRecord(int recordId, byte[] buffer, int offset) 返回存储在由 buffer 代表的字节数组中给定记录的数据。public byte[] getRecord(int recorded)返回由 recordId 代表的数据的副本。public void setRecord(int recordId, byte[] newData, int offset, int numBytes)在 recordId 所代表记录的位置设置新信息，新信息是以 offset 作为它的起始索引，并以 numBytes 作为它的长度的字节流（newData）。

MIDlet 调用 deleteRecord()方法来从记录存储中删除记录。public void deleteRecord(int recordId) 删除由 recordId 代表的记录。这个记录的 recordId 接下来不能重用。

【程序实现】

```
//MultiFieldsMIDlet.java
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
import java.io.*;

public class MultiFieldsMIDlet extends MIDlet implements CommandListener {
    private Command exitCommand;
    private Display display;
    private String[] names = { "张三", "李四", "王五" };

    public MultiFieldsMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("离开", Command.EXIT, 1);
    }

    public void startApp() {
        TextBox aTextBox = new TextBox("主画面", null, 256, TextField.ANY);
        RecordStore rs = null;
        byte[] nameEmail = null;
        boolean existingOrNot = false;
        boolean OK = true;
      //如果存在名为 myRS 的 RecordStore
        existingOrNot = existing("myRS");
        if (existingOrNot) {
            try {
                rs = RecordStore.openRecordStore("myRS", false);
            } catch (Exception e) {
                OK = false;
            } finally {
                if (OK) {
```

```
                aTextBox.setString("myRS已存在并且已打开");
            } else {
                aTextBox.setString("myRS已存在但无法打开");
            }
        }
    } else {
        try {
            rs = RecordStore.openRecordStore("myRS", true);
        } catch (Exception e) {
            OK = false;
        } finally {
            if (OK) {
                aTextBox.setString("myRS虽未存在，但已创建并完成打开的操作！");
            } else {
                aTextBox.setString("myRS虽未存在，但无法创建，打开文件失败！");
            }
        }
    }

    Person2 aPerson = null;
    if (OK) //如果myRS已经存在
        try {
            for (int i = 0; i < names.length; i++) {
                aPerson = new Person2();
                aPerson.write(names[i], "name@163.com", 21);
                byte[] data = aPerson.changeToByteArray();
                int recordID = aPerson.getRecordID();
                if (recordID != -1) {
               //在recordID的位置设置新信息
                    rs.setRecord(recordID, data, 0, data.length);
                } else {
               // 插入新信息，获得recordID后更新Person2中的信息
                    recordID = rs.addRecord(data, 0, data.length);
                    aPerson.setRecordID(recordID);
                }
                aPerson = null;
            }
            aTextBox.setString("已完成新增记录的操作！");
        } catch (Exception e) {
            aTextBox.setString("新增记录的操作失败！");
        }

    String result = "";
    aPerson = new Person2();

    if (OK)
        try {
        //获取RecordStore中的记录数
            int number = rs.getNumRecords();
            byte[] data;
        //输出每个RecordStroe的记录的信息
            for (int i = 1; i < rs.getNextRecordID(); i++) {
                data = rs.getRecord(i);
                aPerson.changeFromByteArray(data);
                result += "第" + i + "笔\n" + "姓名：" + aPerson.getName()
                        + "\n" + "E-mail：" + aPerson.getEMail() + "\n"
                        + "年龄：" + aPerson.getAge() + "\n";
            }

            result += "共有" + number + "笔记录：\n";
            aTextBox.setString(result);
        } catch (Exception e) {
            aTextBox.setString("获取记录的操作失败！");
            try {
                rs.closeRecordStore();
                System.out.println("1.Closed.");
```

```
                    RecordStore.deleteRecordStore("myRS");
                    System.out.println("delete OK");
                } catch (Exception x) {
                }
            } finally {
                try {
                    if (rs != null)
                        rs.closeRecordStore();
                    rs.deleteRecordStore("myRS");
                } catch (Exception e) {
                }
            }

        aTextBox.setString(result);
        aTextBox.addCommand(exitCommand);
        aTextBox.setCommandListener(this);
        display.setCurrent(aTextBox);
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    //    检查指定名字的 RecordStore 是否存在
    public boolean existing(String recordStoreName) {
        boolean existingOrNot = false;
        RecordStore rs = null;
        if (recordStoreName.length() > 32)
            return false;
        try {
            rs = RecordStore.openRecordStore(recordStoreName, false);
        } catch (RecordStoreNotFoundException e) {
            existingOrNot = false;
        } catch (Exception e) {
        } finally {
            try {
                rs.closeRecordStore();
            } catch (Exception e) {
            }
        }
        return existingOrNot;
    }

    public void commandAction(Command c, Displayable s) {
        destroyApp(false);
        notifyDestroyed();
    }
}

class Person2 {
    private int ID = -1;

    private String name;

    private String EMail;

    private int age;

    public void write(String name, String EMail, int age) {
        this.name = name;
        this.EMail = EMail;
        this.age = age;
    }

    public void setRecordID(int ID) {
        this.ID = ID;
    }

    public int getRecordID() {
```

```
        return ID;
    }

    //    将Person2的信息转化为字节数组存储
    public byte[] changeToByteArray() {
        byte[] data = null;
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            DataOutputStream dos = new DataOutputStream(baos);
            dos.writeUTF(name);
            dos.writeUTF(EMail);
            dos.writeInt(age);
            data = baos.toByteArray();

            baos.close();
            dos.close();
        } catch (Exception e) {
        }
        return data;
    }

    //    从字节数组中读取数据
    public void changeFromByteArray(byte[] data) {
        try {
            ByteArrayInputStream bais = new ByteArrayInputStream(data);
            DataInputStream dis = new DataInputStream(bais);

            name = dis.readUTF();
            EMail = dis.readUTF();
            age = dis.readInt();

            bais.close();
            dis.close();
        } catch (Exception e) {
        }
    }

    public String getName() {
        return name;
    }
    public String getEMail() {
        return EMail;
    }

    public int getAge() {
        return age;
    }
}
```

【运行结果】

运行结果如图 13-8 所示。



图 13-8

【分析与总结】

知道了如何在 MIDP 中进行数据存储管理后，可以对上一例添加通讯簿信息的添加、删除和修改操作。

实例 134 使用定时器

本实例演示如何使用 Timer 类执行计划任务。

【编程要点】

定义并定时安排任务执行用到 TimerTask 的 java.util.TimerTask 类是一个抽象类，作为所有其他计划任务的基类；而 java.util.Timer 类创建并管理任务执行的线程。

Timer 类调用 run()方法执行任务。由于某个时刻的每个 Timer 对象只允许一个任务运行，所以该方法必须在完成所做工作后退出。定义了一个任务后，通过创建一个 Timer 实例并调用 schedule()方法计划任务。

【程序实现】

```
//SimpleTimerMIDlet.java
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

public class SimpleTimerMIDlet extends MIDlet {
    Display display;
    StarField field = new StarField();
    FieldMover mover = new FieldMover();
    Timer timer = new Timer();

    public SimpleTimerMIDlet() {
        display = Display.getDisplay(this);
    }
    protected void destroyApp(boolean unconditional) {
    }
    protected void startApp() {
        display.setCurrent(field);
        //在执行前等待 100 毫秒，然后每 100 毫秒执行一次
        timer.schedule(mover, 100, 100);
    }
    protected void pauseApp() {
    }

    public void exit() {
        timer.cancel(); //停止滚动
        destroyApp(true);
        notifyDestroyed();
    }

    class FieldMover extends TimerTask {
        public void run() {
            field.scroll();
        }
    }

    class StarField extends Canvas {
        int height;
        int width;
        int[] stars;
        Random generator = new Random();
        boolean painting = false;

        public StarField() {
            height = getHeight();
            width = getWidth();
```

```
            stars = new int[height];
            for (int i = 0; i < height; ++i) {
                stars[i] = -1;
            }
        }

        public void scroll() {
            if (painting)
                return;
            for (int i = height - 1; i > 0; --i) {
                stars[i] = stars[i - 1];
            }
            stars[0] = (generator.nextInt() % (3 * width)) / 2;
            if (stars[0] >= width) {
                stars[0] = -1;
            }
            repaint();
        }

        protected void paint(Graphics g) {
            painting = true;
            g.setColor(0, 0, 0);
            g.fillRect(0, 0, width, height);
            g.setColor(255, 255, 255);
            for (int y = 0; y < height; ++y) {
                int x = stars[y];
                if (x == -1)
                    continue;
                g.drawLine(x, y, x, y);
            }
            painting = false;
        }

        protected void keypressed(int keycode) {
            exit();
        }
    }
}
```

【运行结果】

运行结果如图 13-9 所示。

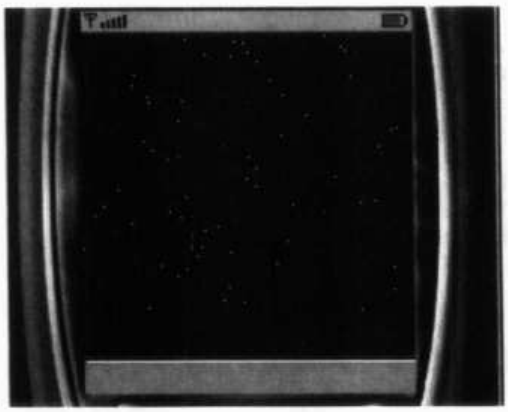


图 13-9

屏幕上不断有白色小点降落。

【分析与总结】

每个 Timer 对象创建和管理一个后台线程，一个单个的应用程序可能只需要一个单个 Timer。创建的 Timer 对象的数量无限制，可以在任何时间停止一个 Timer 并调用其 cancel() 方法结束后台线程。一个被停止的 Timer 不能再次重新启动，必须另外创建一个新的 Timer 并计划任务。另外，Timer 对象可以被不同线程使用，而不需要额外的同步处理。

实例 135 无线 HTTP 连接

本实例演示如何使用 Generic Connection 框架建立 HTTP 连接。

【编程要点】

一个使用 MIDP 的设备必须通过 HTTP 请求与某个网络服务器交互。如果网络不直接支持 HTTP，则设备请求必须通过网关路由。在 MIDP 中可通过 Generic Connection 框架与网络交互，这个框架是一个由 CLDC 定义的代替了大多数 J2SE 中 java.io 和 java.net 类的相关类和接口的集合。

产生一个 HTTP 请求需要使用 Connector.open 方法和一个 URL 并把结果放到 HttpConnection 中，即 HttpConnection conn = Connector.open("http://www.java.sun.com")。

【程序实现】

```
//ConnectHttpMIDlet.java
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;

public class ConnectHttpMIDlet extends MIDlet implements CommandListener {
    private Command exitCommand;
    private Command sendCommand;
    private Command backCommand;
    private Display display;
    private String defaultURL = "http://www.java.sun.com";
    private Form mainForm, resultForm;
    private TextField URL;
    private StringItem resultItem;

    public ConnectHttpMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("离开", Command.EXIT, 1);
        sendCommand = new Command("送出", Command.OK, 1);
        backCommand = new Command("上页", Command.OK, 1);
    }

    public void startApp() {
        URL = new TextField(null, defaultURL, 250, TextField.URL);
        mainForm = new Form("联机数据");
        mainForm.append(URL);
        mainForm.addCommand(sendCommand);
        mainForm.addCommand(exitCommand);
        mainForm.setCommandListener(this);
        display.setCurrent(mainForm);
    }

    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }

    public void commandAction(Command c, Displayable s) {
        if (c == sendCommand) {
            String result = "";
            resultItem = new StringItem(null, result);
            resultForm = new Form("结果");
            String URLString = URL.getString();
            try { //连接 URL，返回字符串信息
                result = requestUsingGET(URLString);
            } catch (IOException e) {
                result = "联机失败";
            }
```

```
                resultItem.setText(result);
                resultForm.append(resultItem);
                resultForm.addCommand(backCommand);
                resultForm.setCommandListener(this);
                display.setCurrent(resultForm);
            } else if (c == backCommand) {
                URL.setString(defaultURL);
                display.setCurrent(mainForm);
            } else {
                destroyApp(false);
                notifyDestroyed();
            }
        }

        //建立Http连接
        private String requestUsingGET(String URLString) throws IOException {
            HttpConnection hpc = null;
            String content = "";
            try {
                hpc = (HttpConnection) Connector.open(URLString);
                content += "URL: " + hpc.getURL() + "\n";
                content += "Protocol: " + hpc.getProtocol() + "\n";
                content += "Host: " + hpc.getHost() + "\n";
                content += "Port: " + hpc.getPort() + "\n";
                content += "File: " + hpc.getFile() + "\n";
                content += "Query: " + hpc.getQuery() + "\n";
                content += "reference: " + hpc.getRef() + "\n";
                content += "request method: " + hpc.getRequestMethod();
            } catch (IOException e) {
                System.out.println("error");
            }
            try {
                if (hpc != null)
                    hpc.close();
            } catch (IOException e2) {
            }
            return content;
        }
    }
```

【运行结果】

调用 requestUsingGET 方法后连接到指定 URL 并获取 HTTP 响应代码。

【分析与总结】

通用连接框架（Generic Connection Framework，GCF）为资源有限的设备提供了一个可扩展的、通用的 I/O 框架。以下是支持 HTTP 的 GCF 类，如图 13-10 所示。

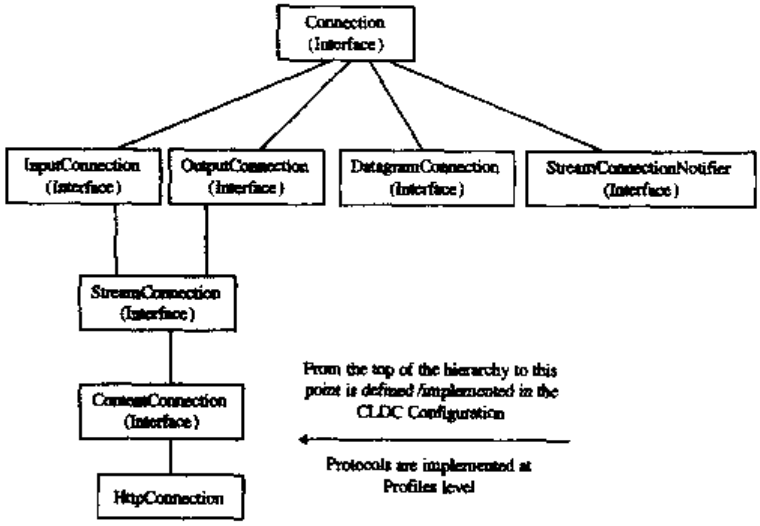


图 13-10

原来的 MIDP 1.0 规范只要求设备支持 HTTP 连接协议，而更新的 MIDP 2.0 规范要求同时支持 HTTP 和 HTTPS，后者提供了对更安全的网络连接的支持。

小结

本章初步介绍了 J2ME 编程的基础。重点是 J2ME 中的事件系统、菜单的设计、记录存储等。

第 14 章　Java 其他编程

本章介绍 Java 的其他方面的编程，包括 JNI 编程、获取系统信息以及介绍 MVC 编程模式的贪食蛇游戏。

实例 136　JNI 调用本地 C 语言

本实例演示如何利用 JNI 调用本地 C 语言。

【编程要点】

Java 以其跨平台的特性深受人们喜爱，而又正由于它的跨平台的原因，使得它和本地机器的各种内部联系变得很少，约束了它的功能。解决 Java 对本地操作的一种方法就是 JNI。

Java 通过 JNI 调用本地方法，而本地方法是以库文件的形式存放的（在 Windows 平台上是 DLL 文件形式，在 Unix 机器上是 SO 文件形式）。通过调用本地的库文件的内部方法，使 Java 可以实现和本地机器的紧密联系，调用系统级的各接口方法。

在 Java 程序中，首先需要在类中声明所调用的库名称，格式如下：

```
static {
   System.loadLibrary("Test");
}
```

在这里，库的扩展名字可以不用写出来，究竟是 DLL 还是 SO，由系统自己判断。还需要对将要调用的方法做本地声明，关键字为 native。并且只需要声明，而不需要具体实现。然后编译该 Java 程序文件，生成 CLASS，再用 javah 命令，JNI 就会生成 C/C++的头文件。然后在编写的 C 文件中引入这个头文件。

【程序实现】

Java 部分的代码如下：

```
//TestDLL.java
public class TestDLL {
    static {
        System.loadLibrary("Test");
    }
    public native static void helloWorld();
    public native static String cTOJava();

    public static void main(String[] args)
    {
        TestDLL test = new TestDLL();
        test.helloWorld();
        System.out.println(test.cTOJava());
    }
}
```

C 实现部分：

刚才用 javah TestDLL 生成的 TestDLL.h 头文件内容如下：

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class TestDLL */

#ifndef _Included_TestDLL
#define _Included_TestDLL
#ifdef __cplusplus
extern "C" {
```

```
#endif
/*
 * Class:     TestDLL
 * Method:    helloWorld
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_TestDLL_helloWorld
  (JNIEnv *, jclass);

/*
 * Class:     TestDLL
 * Method:    cTOJava
 * Signature: ()Ljava/lang/String;
 */
JNIEXPORT jstring JNICALL Java_TestDLL_cTOJava
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

自行编写的 TestDLL.cpp 文件内容如下：

```
#include"TestDLL.h"

JNIEXPORT void JNICALL Java_TestDLL_helloWorld(JNIEnv *env, jclass obj){
    printf("hello world!\n");  //在此处实现 java 类中定义的方法
    return;
}

JNIEXPORT jstring JNICALL Java_TestDLL_cTOJava(JNIEnv *env, jclass obj){
    jstring jstr;
    char str[] = "hello c and java\n";
    jstr = env->NewStringUTF(str);
    return jstr;
}
```

【运行结果】

注意在编译 TestDLL.cpp 文件时要把 jdk include 里面的头文件放到 VC 的库路径中去，编译后生成的 Test.dll 放到 TestDLL.class 的同一目录下，在控制台输入 java TestDLL 后输出如下内容：

```
hello world!
Hello c and Java
```

【分析与总结】

在 TestDLL.cpp 具体实现的时候，往往只关心两个函数原型：JNIEXPORT void JNICALL Java_TestDLL_helloWorld(JNIEnv *env, jclass obj)和 JNIEXPORT jstring JNICALL Java_TestDLL_cTOJava(JNIEnv *env, jclass obj)分别对应 TestDLL.java 文件中的两个函数 public native static void helloWorld()和 public native static String cTOJava()。这里 JNIEXPORT 和 JNICALL 都是 JNI 的关键字，表示此函数是要被 JNI 调用的。而 jstring 是以 JNI 为中介使 Java 的 String 类型与本地的 string 沟通的一种类型。

NewStringUTF()是 JNI 函数，从一个包含 UTF 格式编码字符的 char 类型数组中创建一个新的 jstring 对象。

在 C 和 Java 编程语言之间传送值时，需要理解这些值类型在这两种语言间的对应关系。这些都在头文件 jni.h 中，用 typedef 语句声明了这些类在目标平台上的代价类。

实例 137 获取本机的 Mac 地址

本实例实现如何获取本机的 Mac 地址。

【编程要点】

Runtime 类的 exec(String)方法表示在独立的进程中执行指定字符串命令；getRuntime()方法获得与当前的 Java 应用相关的运行时间对象。

Runtime 的 exec 方法返回 Process 子类的一个实例，该实例可用来控制进程并获取相关信息。它的 getInputStream()方法可获取子进程的输入流。waitFor()方法等待子进程的完成，如果已终止该子进程，此方法立即返回。如果没有终止该子进程，进程的调用将阻塞，直到退出子进程。

【程序实现】

```
//MacAddressAccess.java
import java.io.*;
public class MacAddressAccess {
    public MacAddressAccess() {
    }
    private static String getMyMac() {
        String s = "";
        try {
            String s1 = "ipconfig /all";
            //相当于在命令行下直接使用 ipconfig /all
            Process process = Runtime.getRuntime().exec(s1);
            BufferedReader bufferedreader = new BufferedReader(new
                  InputStreamReader(process.getInputStream()));
            String line = bufferedreader.readLine();
            //筛选出 mac 地址
            for(;line != null;) {
                String nextLine = bufferedreader.readLine();
                if(line.indexOf("Physical Address") > 0){
                    int i = line.indexOf("Physical Address") + 36;
                    s = line.substring(i);
                    break;
                }
                line = nextLine;
            }
            bufferedreader.close();
            process.waitFor();
        } catch(Exception exception) {
            s = "";
        }
        return s.trim();
    }
    public static void main(String[] args){
        //显示 Mac 地址在命令行界面
        System.out.println(MacAddressAccess.getMyMac());
    }
}
```

【运行结果】

控制台输出本机的 Mac 地址。

【分析与总结】

通过执行 config /all 命令得到字符串信息，然后再从中筛选出 Physical Address 的信息。

实例 138　获取 Java 的系统信息

本实例演示如何获取 Java 的系统信息。

【编程要点】

java.lang 包 System 类提供了 getProperty 方法：

public static String getProperty(String key)，获取指定关键字指示的系统属性。

首先，如果有一安全管理器，则用 key 做参数调用它的 checkPropertyAccess 方法，可能返回一个系统异常。

如果没有当前系统属性集合，将生成一个系统属性集合并如同 getProperties 方法那样初始化它。

参数：key -系统属性名。

返回值：系统属性的字符串值，或如果没有那个关键字的属性则返回 null。

抛出：SecurityException，若当前线程不能存取系统属性或指定属性。

【程序实现】

```
//Information.java
public class Information {
    public static void main(String args[]) {
        System.out.println("java_vendor:" + System.getProperty("java.vendor"));
        System.out.println("java_vendor_url:"
                + System.getProperty("java.vendor.url"));
        System.out.println("java_home:" + System.getProperty("java.home"));
        System.out.println("java_class_version:"
                + System.getProperty("java.class.version"));
        System.out.println("java_class_path:"
                + System.getProperty("java.class.path"));
        System.out.println("os_name:" + System.getProperty("os.name"));
        System.out.println("os_arch:" + System.getProperty("os.arch"));
        System.out.println("os_version:" + System.getProperty("os.version"));
        System.out.println("user_name:" + System.getProperty("user.name"));
        System.out.println("user_home:" + System.getProperty("user.home"));
        System.out.println("user_dir:" + System.getProperty("user.dir"));
        System.out.println("java_vm_specification_version:"
                + System.getProperty("java.vm.specification.version"));
        System.out.println("java_vm_specification_vendor:"
                + System.getProperty("java.vm.specification.vendor"));
        System.out.println("java_vm_specification_name:"
                + System.getProperty("java.vm.specification.name"));
        System.out.println("java_vm_version:"
                + System.getProperty("java.vm.version"));
        System.out.println("java_vm_vendor:"
                + System.getProperty("java.vm.vendor"));
        System.out
                .println("java_vm_name:" + System.getProperty("java.vm.name"));
        System.out.println("java_ext_dirs:"
                + System.getProperty("java.ext.dirs"));
        System.out.println("file_separator:"
                + System.getProperty("file.separator"));
        System.out.println("path_separator:"
                + System.getProperty("path.separator"));
        System.out.println("line_separator:"
                + System.getProperty("line.separator"));
    }
}
```

【运行结果】

控制台输出 Java 的生产商、版本等信息，以及操作系统的信息。

实例 139 贪食蛇游戏

本实例利用 MVC 编程技术实现一个常见的贪食蛇游戏。

【编程要点】

模型/界面/控制器（Model/View/Controller，MVC）编程技术允许一个开发者将一个可视化接口连接到一个面向对象的设计中，而同时还可以避免我们上面讨论的几个问题。MVC 最初是为 Smalltalk 语言而设计的。MVC 通过创建下面三个层将面向对象的设计与可

视化接口分开：

模型（Model）：模型包含完成任务所需要的所有的行为和数据。模型一般由许多类组成并且使用面向对象的技术来创建满足五个设计目标的程序。

界面（View）：一个界面就是一个程序的可视化元素，如对话框、选单、工具条等。界面显示从模型中提供的数据，它并不控制数据或提供除显示外的其他行为。一个单一的程序或模型一般有两种界面行为。

控制器（Controller）：控制器将模型映射到界面中。控制器处理用户的输入，每个界面有一个控制器。它是一个接收用户输入、创建或修改适当的模型对象并且将修改在界面中体现出来的状态机。控制器在需要时还负责创建其他的界面和控制器。

控制器一直决定哪些界面和模型组件应该在某个给定的时刻是活动的，它一直负责接收和处理用户的输入，来自用户输入的任何变化都被从控制器送到模型。

界面从模型内的对象中显示数据。这些对象的改变可以通过也可以不通过用户的交互操作来完成。如：在一个 Web 浏览器中负责接收页面的对象收集和装配栈中的信息，必须有某种方式来让这些对象通知界面数据已经被改变了。在模型变化时有两种方法来对界面进行更新。

在第一种方法中，界面可以告诉模型它正在监视哪些对象。当这些对象中有任何一个发生变化时，一个信息就被发送给界面。界面接收这些信息并且相应地进行更新。

第二个方法并不直接将界面连接到模型中，它的控制器负责在模型变化时更新界面。控制器通过对模型对象或观察器方法进行监测来检测模型中的变化。这个方法不用了解界面的模型知识，因此界面就变成是可以跨应用使用的。

MVC 设计允许一个开发者将一个好的面向对象的设计与用户接口隔离开来，允许在同样的模型中容易地使用多个接口，并且允许在实现阶段对接口作大的修改而不需要对相应的模型进行修改。

【程序实现】

主程序 GreedSnake.java 文件内容如下：

```
public class GreedSnake {
    public static void main(String[] args) {
        SnakeModel model = new SnakeModel(20,30);
        SnakeControl control = new SnakeControl(model);
        SnakeView view = new SnakeView(model,control);
        model.addObserver(view);
        (new Thread(model)).start();
    }
}
```

模型类 SnakeModel.java 文件内容如下：

```
import javax.swing.*;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Observable;
import java.util.Random;

/**
 * 游戏的 Model 类，负责所有游戏相关数据及运行
 */
class SnakeModel extends Observable implements Runnable {
    boolean[][] matrix;                          //指示位置上有没蛇体或食物
    LinkedList nodeArray = new LinkedList(); //蛇体
    Node food;
```

```
int maxX;
int maxY;
int direction = 2;                          //蛇运行的方向
boolean running = false;                    //运行状态
int timeInterval = 200;                     //时间间隔，毫秒
double speedChangeRate = 0.75;              //每次得速度变化率
boolean paused = false;                     //暂停标志

int score = 0;                              //得分
int countMove = 0;                          //吃到食物前移动的次数

//UP and DOWN should be even
//RIGHT and LEFT should be odd
public static final int UP = 2;
public static final int DOWN = 4;
public static final int LEFT = 1;
public static final int RIGHT = 3;

public SnakeModel( int maxX, int maxY) {
    this.maxX = maxX;
    this.maxY = maxY;

    reset();
}

public void reset(){
    direction = SnakeModel.UP;              //蛇运行的方向
    timeInterval = 200;                     //时间间隔，毫秒
    paused = false;                         //暂停标志
    score = 0;                              //得分
    countMove = 0;                          //吃到食物前移动的次数

    //initial matirx, 全部清 0
    matrix = new boolean[maxX][];
    for (int i = 0; i < maxX; ++i) {
        matrix[i] = new boolean[maxY];
        Arrays.fill(matrix[i], false);
    }

    //初始化蛇体，如果横向位置超过 20 个，长度为 10，否则为横向位置的一半
    int initArrayLength = maxX > 20 ? 10 : maxX / 2;
    nodeArray.clear();
    for (int i = 0; i < initArrayLength; ++i) {
        int x = maxX / 2 + i;
        int y = maxY / 2;
        nodeArray.addLast(new Node(x, y));
        matrix[x][y] = true;
    }

    //创建食物
    food = createFood();
    matrix[food.x][food.y] = true;
}

public void changeDirection(int newDirection) {
    //改变的方向不能与原来方向同向或反向
    if (direction % 2 != newDirection % 2) {
        direction = newDirection;
    }
}

/**
 * 运行一次
 * @return
 */
public boolean moveOn() {
```

```
        Node n = (Node) nodeArray.getFirst();
        int x = n.x;
        int y = n.y;

        //根据方向增减坐标值
        switch (direction) {
            case UP:
                y--;
                break;
            case DOWN:
                y++;
                break;
            case LEFT:
                x--;
                break;
            case RIGHT:
                x++;
                break;
        }

        //如果新坐标落在有效范围内，则进行处理
        if ((0 <= x && x < maxX) && (0 <= y && y < maxY)) {
            if (matrix[x][y]) {                          //如果新坐标的点上有东西（蛇体或者食物）
                if (x == food.x && y == food.y) {        //吃到食物，成功
                    nodeArray.addFirst(food);            //从蛇头增长

                    //分数规则，与移动改变方向的次数和速度两个元素有关
                    int scoreGet = (10000 - 200 * countMove) / timeInterval;
                    score += scoreGet > 0 ? scoreGet : 10;
                    countMove = 0;

                    food = createFood();                 //创建新的食物
                    matrix[food.x][food.y] = true;       //设置食物所在位置
                    return true;
                } else                                   //吃到蛇体自身，失败
                    return false;
            } else {                                     //如果新坐标的点上没有东西（蛇体），移动
蛇体
                nodeArray.addFirst(new Node(x, y));
                matrix[x][y] = true;
                n = (Node) nodeArray.removeLast();
                matrix[n.x][n.y] = false;
                countMove++;
                return true;
            }
        }
        return false;                                    //触到边线，失败
    }

    public void run() {
        running = true;
        while (running) {
            try {
                Thread.sleep(timeInterval);
            } catch (Exception e) {
                break;
            }

            if (!paused) {
                if (moveOn()) {
                    setChanged();                        //Model 通知 View 数据已经更新
                    notifyObservers();
                } else {
                    JOptionPane.showMessageDialog(null,
                            "you failed",
                            "Game Over",
                            JOptionPane.INFORMATION_MESSAGE);
```

```
                    break;
                }
            }
        }
        running = false;
    }

    private Node createFood() {
        int x = 0;
        int y = 0;
        //随机获取一个有效区域内的与蛇体和食物不重叠的位置
        do {
            Random r = new Random();
            x = r.nextInt(maxX);
            y = r.nextInt(maxY);
        } while (matrix[x][y]);

        return new Node(x, y);
    }

    public void speedUp() {
        timeInterval *= speedChangeRate;
    }
    public void speedDown() {
        timeInterval /= speedChangeRate;
    }

    public void changePauseState() {
        paused = !paused;
    }

    public String toString() {
        String result = "";
        for (int i = 0; i < nodeArray.size(); ++i) {
            Node n = (Node) nodeArray.get(i);
            result += "[" + n.x + "," + n.y + "]";
        }
        return result;
    }
}

class Node {
    int x;
    int y;

    Node(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

视图类 SnakeView.java 文件内容如下：

```
import javax.swing.*;
import java.awt.*;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Observable;
import java.util.Observer;

/**
 * MVC 模式中得 Viewer，只负责对数据的显示，而不用理会游戏的控制逻辑
 */
public class SnakeView implements Observer {
    SnakeControl control = null;
    SnakeModel model = null;

    JFrame mainFrame;
    Canvas paintCanvas;
    JLabel labelScore;
```

```
public static final int canvasWidth = 200;
public static final int canvasHeight = 300;

public static final int nodeWidth = 10;
public static final int nodeHeight = 10;
public SnakeView(SnakeModel model, SnakeControl control) {
    this.model = model;
    this.control = control;
    mainFrame = new JFrame("GreedSnake");
    Container cp = mainFrame.getContentPane();

    //创建顶部的分数显示
    labelScore = new JLabel("Score:");
    cp.add(labelScore, BorderLayout.NORTH);

    //创建中间的游戏显示区域
    paintCanvas = new Canvas();
    paintCanvas.setSize(canvasWidth + 1, canvasHeight + 1);
    paintCanvas.addKeyListener(control);
    cp.add(paintCanvas, BorderLayout.CENTER);

    //创建底下的帮助栏
    JPanel panelButtom = new JPanel();
    panelButtom.setLayout(new BorderLayout());
    JLabel labelHelp;
    labelHelp = new JLabel("PageUp, PageDown for speed;", JLabel.CENTER);
    panelButtom.add(labelHelp, BorderLayout.NORTH);
    labelHelp = new JLabel("ENTER or R or S for start;", JLabel.CENTER);
    panelButtom.add(labelHelp, BorderLayout.CENTER);
    labelHelp = new JLabel("SPACE or P for pause", JLabel.CENTER);
    panelButtom.add(labelHelp, BorderLayout.SOUTH);
    cp.add(panelButtom, BorderLayout.SOUTH);

    mainFrame.addKeyListener(control);
    mainFrame.pack();
    mainFrame.setResizable(false);
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainFrame.setVisible(true);
}

void repaint() {
    Graphics g = paintCanvas.getGraphics();

    //画背景
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, canvasWidth, canvasHeight);

    //画蛇体
    g.setColor(Color.BLACK);
    LinkedList na = model.nodeArray;
    Iterator it = na.iterator();
    while (it.hasNext()) {
        Node n = (Node) it.next();
        drawNode(g, n);
    }

    //画食物
    g.setColor(Color.RED);
    Node n = model.food;
    drawNode(g, n);
    updateScore();
}

private void drawNode(Graphics g, Node n) {
    g.fillRect(n.x * nodeWidth,
            n.y * nodeHeight,
            nodeWidth - 1,
            nodeHeight - 1);
}
```

```
    public void updateScore() {
        String s = "Score: " + model.score;
        labelScore.setText(s);
    }

    public void update(Observable o, Object arg) {
        repaint();
    }
}
```

控制器类 SnakeControl.java 文件内容如下：

```
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * MVC 中的 Controler，负责接收用户的操作，并把用户操作通知 Model
 */
public class SnakeControl implements KeyListener{
    SnakeModel model;

    public SnakeControl(SnakeModel model){
        this.model = model;
    }

    public void keyPressed(KeyEvent e) {
        int keyCode = e.getKeyCode();
        if (model.running){                    //运行状态下，处理的按键
            switch (keyCode) {
                case KeyEvent.VK_UP:
                    model.changeDirection(SnakeModel.UP);
                    break;
                case KeyEvent.VK_DOWN:
                    model.changeDirection(SnakeModel.DOWN);
                    break;
                case KeyEvent.VK_LEFT:
                    model.changeDirection(SnakeModel.LEFT);
                    break;
                case KeyEvent.VK_RIGHT:
                    model.changeDirection(SnakeModel.RIGHT);
                    break;
                case KeyEvent.VK_ADD:
                case KeyEvent.VK_PAGE_UP:
                    model.speedUp();
                    break;
                case KeyEvent.VK_SUBTRACT:
                case KeyEvent.VK_PAGE_DOWN:
                    model.speedDown();
                    break;
                case KeyEvent.VK_SPACE:
                case KeyEvent.VK_P:
                    model.changePauseState();
                    break;
                default:
            }
        }

        //任何情况下处理的按键，按键导致重新启动游戏
        if (keyCode == KeyEvent.VK_R ||
                keyCode == KeyEvent.VK_S ||
                keyCode == KeyEvent.VK_ENTER) {
            model.reset();
        }
    }

    public void keyReleased(KeyEvent e) {
    }

    public void keyTyped(KeyEvent e) {
    }
```

}

【运行结果】

运行结果如图 14-1 所示。



图 14-1

【分析与总结】

MVC 通过以下三种方式消除与用户接口和面向对象的设计有关的绝大部分困难：

（1）控制器通过一个状态机跟踪和处理面向操作的用户事件。这允许控制器在必要时创建和破坏来自模型的对象，并且将面向操作的拓扑结构与面向对象的设计隔离开来。这个隔离有助于防止面向对象的设计走向反面。

（2）MVC 将用户接口与面向对象的模型分开。这允许同样的模型不用修改就可使用许多不同的界面显示方式。除此之外，如果模型更新由控制器完成，那么界面就可以跨应用再使用。

（3）MVC 允许应用的用户接口进行大的变化而不影响模型。每个用户接口的变化将只需要对控制器进行修改，但是既然控制器包含很少的实际行为，它是很容易修改的。

面向对象的设计人员在将一个可视化接口添加到一个面向对象的设计中时必须非常小心，因为可视化接口的面向操作的拓扑结构可以大大增加设计的复杂性。

小结

本章是对 Java 编程的补充，涉及 Java 技术的编程非常之广，本章主要介绍了 JNI 的调用，以及获取系统信息的方法。另外还介绍了基于 MVC 模式的贪食蛇游戏，MVC 模式是现在比较流行的编程模式，因为它进一步地简化了程序的复杂性，比如现在流行的 structs 就是 MVC 的开发框架，有兴趣的读者可以进一步地学习这方面的内容。

第 15 章　Java 工具和服务器配置与集成

本章通过 11 个实例介绍了 Java 的各种服务器和开发环境的配置与集成，包括 J2SDK 的安装和配置、第三方包的使用、Appche、Tomcat、Jboss、WebLogic 的安装和配置，Tomcat 和 Apache、Tomcat 和 IIS 的整合，以及开发环境 Jbuilder 集成 WebLogic，Eclipse 开发 J2EE 配置，J2ME 开发环境的配置。

实例 140　J2SE SDK 的安装和配置

本实例演示如何安装和配置 J2SE 程序开发必需的 jdk。

1. 下载

可以从 Sun 公司的主页 http://java.sun.com/j2se/1.4.2/download.html 下载所需要的版本 j2sdk1.4.2。

2. 安装步骤

执行 j2sdk-1_4_2-windows-i586.exe，安装到指定目录，这里是 C:\j2sdk1.4.2。注意保证 jdk 安装路径中不能有空格。

3. 配置

以 winxp 系统为例。

（1）配置 Path 变量，该变量在系统执行 jdk 命令时指定执行程序的位置并执行，如常用的 javac 和 java。

右击“我的电脑”→“属性”→“高级”→“环境变量”，如果存在 Path 变量，就只需添加变量值“C:\j2sdk1.4.2\bin;”，注意前后用分号隔开，如果不存在 Path 变量，就在“系统变量”中单击“新建”按钮，弹出“新建系统变量”对话框，输入变量名“Path”和变量值“C:\j2sdk1.4.2\bin;”。

（2）再用同样的方法设置 CLASSPATH 环境变量，该变量指定 Java 程序编译和运行时所需要的 Java 标准类库及其他所需的类文件所在位置，添加变量名“CLASSPATH”和变量值“C:\j2sdk1.4.2\lib\Tools.jar; C:\j2sdk1.4.2\lib\dt.jar”。

（3）单击“确定”按钮完成设置。

4. 运用

j2sdk 软件包包含 Sun 的 java 开发环境。用于开发 java 程序和运行 java 程序时提供运行环境。它还为浏览器提供插件使其可以打开 java 的程序。

j2sdk 包括 appletviewer，extcheck，idlj，jar，jarsigner，java，javac，javadoc，javah，javap，jdb，keytool，native2ascii，orbd，policytool，rmic，rmid，rmiregistry，rmiregistry，serialver，servertool 和 tnameserv。

appletviewer：在浏览器外运行 Java 的 applet 程序。

extcheck：检查特殊 jar 文件的标题和版本是否与在 JDK 软件中的任何的扩展名冲突。

idlj：从已给的 IDL 文件中生成 Java 的联编文件。

jar：合并多个文件为一个 JAR 档案文件。

jarsigner：在 JAR（Java ARchive）文件留下标记，并且确认标记过的 JAR 文件签名以及完整性。

java：通过启动一个 Java 的运行环境来运行一个 Java 的应用程序,加载一个特殊的类（class）并且调用它的主方法（method）。

javac：读用 Java 语言编写的 java 类（class）和接口程序，把它们编译成字节码文件。

javadoc：解析一系列 Java 源码文件的声明和文档注释并生成一系列相关的描述类（class），接口程序（interfaces），构造符（constructors），方法（methods）和域（fields）的 HTML 文件。

javah：为了实现原始方法，生成所需的 C 头文件和源码。

javap：反汇编 Java 的 class 文件。

jdb：是一个简单的命令行的错误调试程序，用于 Java class 文件。

keytool：是钥匙和认证的管理工具。

native2ascii：把含有解码不支持字符的文件转换为用 Latin-1 或 Unicode-encoded 字符的文件。

orbd：用在 CORBA 环境中，使客户端能透明地定位和调用服务器的永久对象。

policytool：图形化的建立和管理策略（policy）文件。

rmic：生成 stub 和 skeleton 类文件，从编译后的包含远程对象实现的 Java 类的名字生成类文件。

rmid：运行激活系统守护进程。

rmiregistry：在当前主机的指定端口创建和运行远程对象注册。

serialver：为一个或多个类返回易于拷贝到进化类中的 serialVersionUID。

servertool：为应用程序注册、取消注册、启动和停止服务器提供了方便的接口。

tnameserv：启动 Java IDL 名称服务器。

可以编写一个简单的 Java 程序检测配置是否成功，可按本书的第一个例子介绍的步骤进行。

实例 141　使用第三方包

本实例介绍如何使用第三方包。Java 虚拟机按以下顺序搜索并加载所有需要的类。

（1）引导类：组成 Java 平台的类，包含 rt.jar 和 i18n.jar 中的类。

（2）扩展类：使用 Java 扩展机制的类，都是位于扩展目录即 jdk 目录下的 jre/lib/ext 中的 jar 档案包。

（3）用户类：自定义类或未使用 Java 扩展机制的第三方产品。必须在 CLASSPATH 环境变量中添加这些类的位置。

用户只需指定用户类的位置，引导类和扩展类是自动查找的。如对于常用的 JDBC Driver，编写 servlet 需要的 servlet 包和 Jmail 等。

如果用户使用 IDE 的话，只需在 IDE 的类路径设置中加入要是用的第三方包的位置即可。

如在 Eclipse 中可以如图 15-1 所示进行设置。

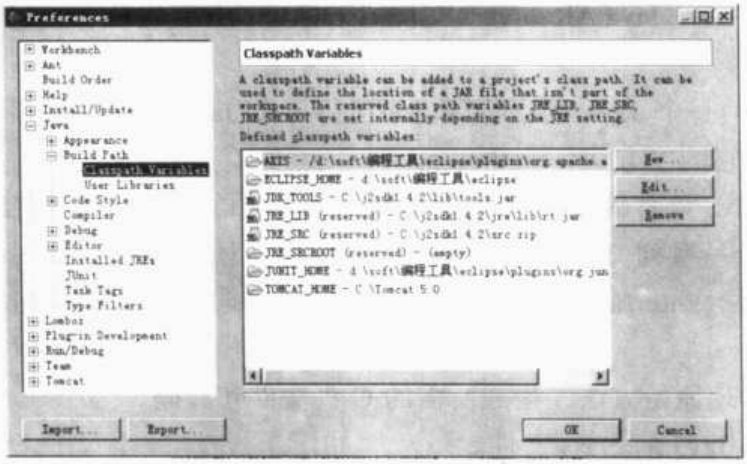


图 15-1

在环境变量中添加 CLASSPATH，然后即可编译运行。以 HelloWorld.class 为例，如果需要在根目录中运行，则在根目录下执行：

```
C:\java HelloWorld
```

如果 CLASSPATH 中没有添加 HelloWorld.class 所在的目录，那么就会报错：Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorld。添加后再运行就可以成功。在直接运行其中的类或者 import 其中的某些类时，也可以用同样的方法。

实例 142 Apache 安装和配置

本实例说明如何安装和配置 Apache。

1. 下载

Windows 平台的 Apache 可以在 http://ajax.apache.org/dist/httpd/binaries/win32/ Apache 上进行下载。这个版本目前有两个分支：1.3.x 和 2.0.x。初学者用 1.3.x 就可以了。本文以 1.3.x 为例。在上面的网址里，包含了两个分支的最新版本。请选择后缀为“exe”或“msi”的安装文件下载。

2. 安装

运行下载的 Apache 文件，即可开始安装。Apache 的安装过程很简单。只需在如图 15-2 所示的窗口中进行设置。

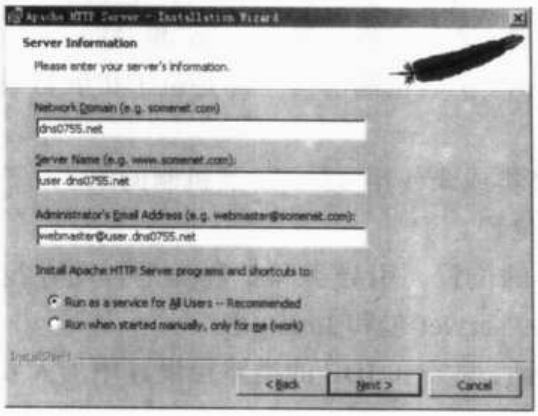


图 15-2

在“Network Domain”里输入域。如果用本站的二级域名，例如“user.dns0755.net”，请输入“dns0755.net”。如果用顶级域名，请输入“xxx.com”。

在“Server Name”里输入服务器名。如果用本站的二级域名，请输入域名“user.dns0755.net”。如果用顶级域名，请输入“www.xxx.com”。

在“Administrator's Email Address”里输入网站管理员的 E-mail 地址。

除此之外，安装过程里所有的选项，全部用默认选项就可以了。

3. 运行

1）Windows 9x

Apache 安装后，会在“开始”→“程序”里增加一个“Apache HTTP Server”的组。进入该组，选择“Control Apache Server”→“Start”即可启动 Apache。

2）Windows NT/2000/XP/2003

在这些系统里，用默认选项安装的 Apache，除了在“开始”→“程序”里增加一个“Apache HTTP Server”的组之外，还会在系统的服务里增加一个 Apache 服务。该服务被设置为系统启动时自动运行。

4. 配置 Apache

1）配置文件

Apache 是一个后台运行的程序，没有界面。所有的配置，都包含在配置文件里。主配置文件是：

C:\Program Files\Apache Group\Apache\conf\httpd.conf

如果要修改 Apache 的配置，可以用任何一个文本编辑工具（例如记事本）编辑这个配置文件。

在配置文件里，以“#”开头的行是注释行。

2）配置选项

在配置文件里的主要选项如下：

```
#Listen 3000
#Listen 12.34.56.78:80
```

Apache 的 IP 地址和端口。一般来说不需要设置，Apache 会绑定在本机所有 IP 地址的 80 端口上。

```
Port: 80
```

Apache 的端口。默认值是 80。

```
ServerName user.dns0755.net
```

Web 服务器的名字。安装时输入的“Server Name”就是保存在这里。

```
DocumentRoot "C:/Program Files/Apache Group/Apache/htdocs"
```

这是网站的根目录。如果想把网站文件存放在“D:\myweb”目录下，可以把 DocumentRoot 后面的目录修改为“D:/myweb”。

```
Options Indexes FollowSymLinks MultiViews
```

为了安全起见，请把上面的“Indexes”删掉。否则别人可以浏览到网站里的所有文件。

```
DirectoryIndex index.html
```

默认首页文件名。在浏览器里输入一个地址（例如 http://user.dns0755.net/）的时候，Apache 会查找这个默认的首页文件打开。如果要配置多个默认首页文件名，请用空格分开。

```
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache/cgi-bin/"
```

CGI 文件存放路径。

3）高级配置

（1）设置虚拟目录。

在配置文件中找下面这行：

```
<IfModule mod_alias.c>
```

在这行下面添加如下内容：

```
Alias /vdir/ "c:/comexe/"
<Directory "c:/comexe/">
Options MultiViews
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

这样，通过访问 http://user.dns0755.net/vdir/，就可以访问到 c:\comexe\目录下的内容。

Apache 默认开设了两个虚拟目录："/icons/" 和 "/manual/"。这两个虚拟目录没有用，可以在配置文件中删除。

（2）如何解决中文网页显示乱码。

① 在配置文件中找包含 "AddLanguage" 或 "AddCharset" 的行，在这些行最前面增加一行：

```
AddDefaultCharset GB2312
```

② 养成良好的习惯，在每个网页的<head></head>里加入这行：

```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
```

一般的中文版网页编辑工具（例如 FrontPage、Dreamweaver 等）都会自动加上这行。

（3）如何解决中文文件名无法访问。

在 IE 菜单栏中选择 "工具" → "internet 选项" → "高级" → "取消" → "总是以 UTF-8 发送 URL"。

这个方法很麻烦，请尽量不要使用中文的网站文件名。

实例 143 Tomcat 的安装和配置

本实例说明如何安装和配置 Tomcat。

1. 下载

在 http://ajax.apache.org/dist/jakarta/上可以选择各种版本的 Tomcat 进行下载。本例是 5.0.x 的版本。

2. 安装

在安装 tomcat 之前必须确认 JDK1.4 以后的版本。解压缩下载的软件包，执行 jakarta-tomcat-5.0.27.exe，安装到指定目录，如 C:\Tomcat 5.0。

按照安装向导完成 Tomcat 的安装，Tomcat 安装目录下的目录结构描述如下：

bin：包含启动/关闭脚本。

conf：包含不同的配置文件，包括 server.xml（Tomcat 的主要配置文件）和为不同的 Tomcat 配置的 web 应用设置缺省值的文件 web.xml。

doc：包含各种 Tomcat 文档。

lib：包含 Tomcat 使用的 jar 文件。Unix 平台此目录下的任何文件都被加到 Tomcat 的 classpath 中。

logs：Tomcat 摆放日志文件的地方。

src：ServletAPI 源文件，这些只有些必须在 Servlet 容器内实现的空接口和抽象类。

webapps：包含 web 项目示例。

3. 设置环境变量

按照前面介绍的方法设置环境变量 JAVA_HOME 为安装 j2sdk 的目录，设置 CATALINA_HOME 为安装 tomcat 的目录，即本实例中的 C:\Tomcat 5.0 目录。

4. 运行

1）启动 Tomcat 服务器

执行 Tomcat 安装目录\bin 下的 startup.bat 文件，启动 Tomcat 5.0，这时 DOS 窗口会显示启动的信息；或者打开“开始”菜单中 Apache Tomcat5.0 的 Monitor Tomcat，在右边的任务栏中会出现 Apache Tomcat 的图标，如果是红色，说明 Tomcat 未启动，这时只需右击该图标，然后选择 start service 来启动 Tomcat，启动后图标变成绿色。

2）测试 Tomcat 是否安装成功

打开 IE 浏览器，在地址栏中输入 http://localhost:8080/。如果显示如图 15-3 所示的页面，则说明安装配置 Tomcat 服务器成功。

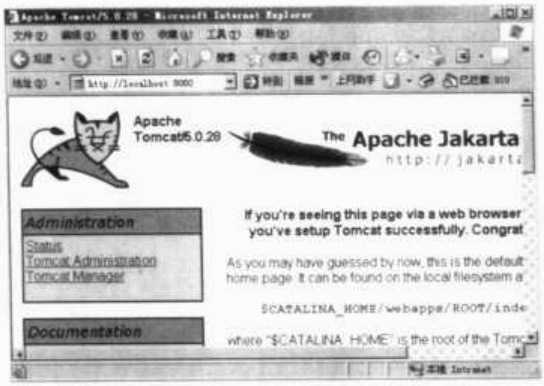


图 15-3

3）关闭 Tomcat

运行 mcat 安装目录\bin 下 shutdown.bat 可以将服务器关闭，或者可以右击图标，选择 exit 就可以退出服务器。

实例 144　整合 Tomcat 和 Apache

本实例介绍 tomcat 5.0.27+Apache 2.0.49+j2sdk 1.4.2+mod_jk2 2.0.4 在 Windows XP 下整合。

安装完 Apache 和 Tomcat 之后定义两个环境变量：%APACHE_HOME% 为 apache 的安装目录，%CATALINA_HOME%为 tomcat 的安装目录。

整合的整个过程中不需要设置环境变量。假设 tomcat 安装服务端口是 8080，apache 安装服务端口是 80。

需要下载 jk2 连接器：mod_jk2.so，下载网址：http://219.217.8.91/jk2.rar。

（1）拷贝 mod_jk2.so 到%APACHE_HOME%\modules。

（2）确信 mod_jk2.so 可读（对于 system 用户，文件属性?>安全）。

（3）创建 jk2.properties 文件，文件具体内容如下：

```
#
# Socket configuration
#
```

```
handler.list=request,container,channelSocket
#
# socket configuration
#
channelSocket.port=8009
channelSocket.address=127.0.0.1
channelSocket.maxPort=port+10
```

（4）把这个文件放入%CATALINA_HOME%\conf，如果已经存在此文件，先备份原来的 jk2.properties。

（5）确信这个文件可读（对于 System 用户）。

（6）创建 workers2.properties 文件，文件具体内容如下：

```
#
# basic workers2.properties file
#
#
# shared memory
#
[shm]
file=Apache 安装路径/logs/shm.file
# ~~~~~~~~~~~~~~
#需要改动的地方，填写 apache 安装的绝对路径
#比如 C:/Program Files/Apache Group/Apache2/
size=1048576
[workerEnv:]
info=Global server options
timing=1
debug=0
#
# IP socket channel
#
[channel.socket:localhost:8009]
port=8009
host=127.0.0.1
#
# define the ajp13 worker
#
[ajp13:localhost:8009]
channel=channel.socket:localhost:8009
#
# Uri mapping - jsp files only
#下面使用的是根目录/*.jsp，可以更改
[uri:/*.jsp]
worker=ajp13:localhost:8009
#
# Uri mapping - all servlets
#可以更改 servlet
[uri:/*]
workers=ajp13:localhost:8009
```

（7）把这个文件放入%APACHE_HOME%\conf。

（8）确信 workers2.properties 文件可读（对 system 用户）。

（9）创建 shm.file 文件。下面提供一个方法去创建 shm.file。命令行方式进入%APACHE_HOME%/logs，输入 copy con: > shm.file 后按回车键。可多次按回车键来输入一些空行，然后按 Ctrl+Z 再按回车键退出。

（10）确信 shm.file 在%APACHE_HOME%/logs/目录里，确信 shm.file 文件可写。

（11）在%APACHE_HOME%\conf\httpd.conf 中添加加载 mod_jk2.so 信息，在此文件中会看到一堆 LoadModule（可以查找 LoadModule）的加载模块，在此处添加如下信息：LoadModule jk2_module modules/mod_jk2.so。

（12）继续在此文件中（%APACHE_HOME%\conf\httpd.conf）去映射 tomcat 的信息，

这将会允许 apache 解析静态的信息（可以查找 Alias，把下面这一行放在 Alias/icons/..... 前后，其实具体位置没什么限制，只是为了以后更改寻找带来方便）：

```
Alias / "Tomcat 安装目录/webapps/"
# ~~~~~~~~~~~~~~~
#Note:
#这里填写 tomcat 安装的绝对路径
#如果把 tomcat 目录\server.xml 中应用程序映射的路径改了的话，这里
#还要做相应得变化
```

（13）关闭 apache 服务器。

（14）关闭 tomcat 服务器。

（15）重新启动 tomcat 服务器。

（16）通过浏览 http://localhost:8080 来测试 tomcat 是否启动（http://不要省略）如果使用 localhost 发现不行的话，换成用户的 IP。

（17）启动 apache 服务器。

（18）测试下面的链接：

http://localhost，打开页面中左侧的链接 Examples——>JSP Examples。

（19）比较下面的链接：

http://localhost:8080 打开页面中左侧的的链接 Examples——>JSP Examples。

（20）分别运行几个 JSP Examples 中的例子，如果运行结果一样的话，说明 tomcat 和 apache 已经整合在一起了。

实例 145　整合 Tomcat 和 IIS

本实例说明如何整合 Tomcat 与 IIS。

IIS（Internet Information Server）是 Microsoft 公司提供的一种功能强大的 Web 服务器，能够很好地支持 ASP 网站。

随着 JSP 技术的成熟和普遍应用，IIS 如何支持 JSP 技术成为一个突出的问题，一个解决方案是以 Tomcat 作为插件插入 IIS 的方式实现。

在 XP 系统下，先安装 Tomcat 5.0 和 IIS6，然后到 http://apache.linuxforum.net/dist/jakarta/tomcat-connectors/jk2/binaries/win32/jakarta-tomcat-connectors-jk2.0.2-win32-IIS.zip 去下载 JK2，由 JK2 负责 IIS 和 Tomcat 的通讯，解开后得到 isapi_redirector2.dll，把它放在了 C:\tomcat\iis 下（目录随便）。

接下来在 Tomcat 安装目录下的 conf 目录下建一个 workers2.properties 文件，把下面这段内容拷进去保存。注意 file=".."之中的路径要用“/”。

```
[shm]
file=c:/Tomcat/logs/jk2.log
size=1048576
# 实际运用时把所有的 localhost 和 8009 换成你自己的 IP 和端口
[channel.socket:localhost:8009]
port=8009
host=localhost
# define the worker
[ajp13:localhost:8009]
channel=channel.socket:localhost:8009
# Uri mapping
[uri:/*.jsp]
[uri:/web/*]
worker=ajp13:localhost:8009
```

```
# define the worker
[status:status]
# Uri mapping
[uri:/jkstatus/*]
worker=status:status
```

之后编辑 jk2.properties 文件，确认其中是否有此一句：

```
request.tomcatAuthentication=false
```

接下来开始编辑注册表，直接建一个 iis.reg 文件，把下面一段拷进去保存然后双击 iis.reg 把信息导入注册表（注意，把涉及的 Tomcat 路径改成用户的 Tomcat 的路径）。

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi
Redirector\2.0]
"serverRoot"="c:\\Tomcat"
"extensionUri"="/jakarta/isapi_redirector2.dll"
"workersFile"="c:\\Tomcat\\conf\\workers2.properties"
"logLevel"="INFO"
```

然后打开“我的电脑”→“控制面板”→“管理工具”→“Internet 信息服务管理器”，在里面选择“网站”中的默认网站（默认网站这里假定指向 C:\web），在其上单击右键选“属性”，在弹出的页框里选择“ISAPI 筛选器”，单击“添加”，筛选器名称输入 jakarta，可执行文件中填入 isapi_redirector2.dll 的绝对路径。“确定”后单击“确定”关闭属性页。再在“默认网站”上单击右键，选择“新建”→“虚拟目录”，别名输入 jakarta，在下一步中路径选为 isapi_redirector2.dll 存放的目录，这里是 C:\tomcat\iis，在下一步的权限中把“执行”项选中。

IIS5 到这里就完成了，IIS6 还要进行以下步骤：

在“Internet 信息服务管理器”的“网站”中的“Web 服务扩展”上单击右键选择“添加一个新的 Web 服务扩展”，弹出的框中扩展名输入 Tomcat，“要求的文件”选“添加”，然后找到 isapi_redirector2.dll 后单击“确定”。在“设置扩展状态为允许”也打上勾，单击“确定”配置完成。

现在启动 Tomcat5，然后 net start w3svc 启动 IIS6。输入 http://localhost:8080/hello.jsp（或其他 jsp 文件）看看执行结果。然后再看看 http://localhost/hello.jsp，如果两者执行结果一样，则说明整合成功。

实例 146 WebLogic 的安装和配置

本实例演示 winxp 下 WebLogic Server 8 安装与配置。

安装过程比较简单，执行 platform812_zh_CN_win32.exe 文件，开始安装。安装目录可自选，这里是在 d:\bea，选择完全安装，其他按默认值就行。下面主要看看配置过程，以下步骤按照 WebLogic 配置出现界面：

（1）“开始”→“程序”→“Bea WebLogic Platform 8.1”→“Configuration Wizard”，启动 Configuration Wizard 工具。

（2）选择“新建 WebLogic 配置”，单击“下一步”。

（3）选择“Basic WebLogic Server Domain”，单击“下一步”。

（4）选择“自定义”，单击“下一步”。

（5）更改 Name，这里设为 myname，单击“下一步”。

（6）选择“否”，单击“下一步”。

（7）选择“否”，单击“下一步”。

（8）选择“否”，单击“下一步”。

（9）输入“User name”和“password”，单击“下一步”，密码需要设置 8 位。

（10）选择作为 Window 服务，单击“下一步”。

（11）默认设置，单击“下一步”。

实例 147　整合 Jbuilder 与 WebLogic

按以下步骤进行：

（1）单击主菜单“Tools”→“Configure Servers”，在旁边选中 WebLogic Platform Server 8.x，然后在右边的 Server Settings 下的 Enable server 打上勾，如图 15-4 所示。

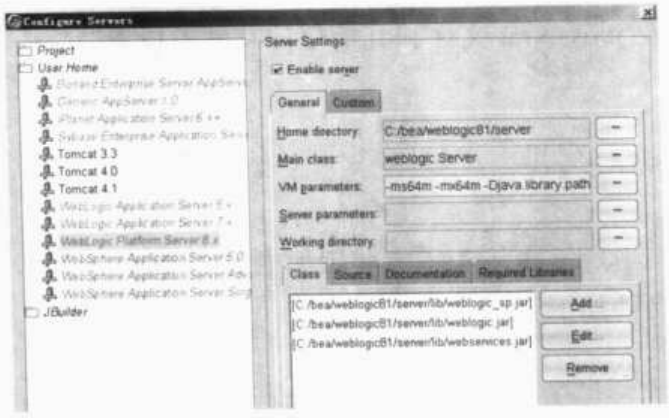


图 15-4

（2）在 General 面板上，“Home Directory:”选择 weblogic server 的安装目录，这里是 C:\bea\weblogic81\server，“Working directory:”选择 C:\bea\weblogic81\server。

（3）切换到 Custom 面板，进行如下设置。“BEA home directory:”选择 C:\bea。“JDK install directory:”选择 Jbuilder 的 jdk 目录，这里是 C:\JBuilder9\jdk1.4。“Domain directory:”选择 C:\bea\user_projects\domains\mydomain。Username 和 Password 填写在配置 Weblogic 里的 mydomain 时的用户名和密码。“Server nmae:”为 myserver，“Listen address:”为 192.168.0.1/本机 IP，其他不变，单击“OK”保存设置，如图 15-5 所示。

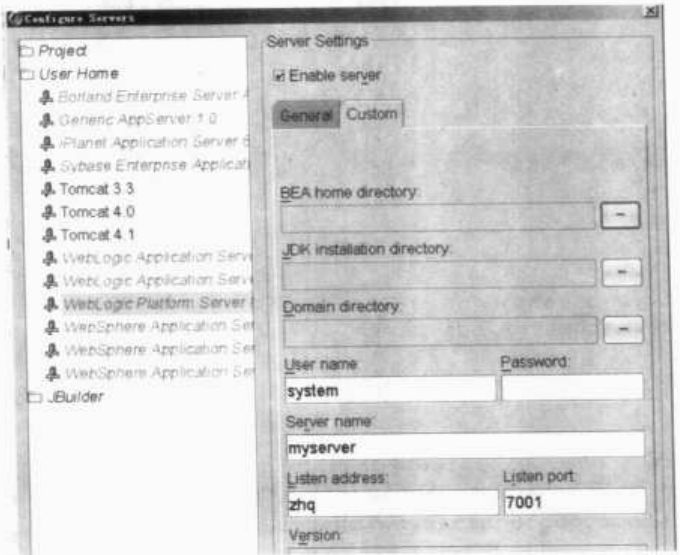


图 15-5

实例 148 Jboss 的安装与配置

本实例演示如何安装 Jboss 与部署 EJB。

可以从 http://www.jboss.org 中下载各种版本的 JBoss，Jboss 的安装很简单，只需要解压到指定的目录就可以了，然后在系统的环境变量中设置 JBOSS_HOME 变量运行 bin 目录下的 run.bat 可以启动服务器。

JBoss 容器中部署各类 EJB 对象的两种方法。下面的演示以 Session Bean 为例，客户端是一个本机上的 Web 应用。这里不对演示程序进行具体介绍。

第一种方式：

把开发好的 ejb.jar 包和 Web 应用的 war 包公共放在 deploy 目录下。

【演示 1】：不需要 jboss.xml 文件和其他任何特殊的设置。

```
ejb-jar.xml:
<ejb-jar>
……
<ejb-name>ejbtest</ejb-name>
……
</ejb-jar>
```

客户端：Web 应用中对 web.xml 不需要添加 EJB 信息，不需要 jboss-web.xml 文件。引用 EJB 对象的 JSP 文件如下：

```
……
InitialContext ctx = new InitialContext();
Object objref=ctx.lookup("ejbtest");// 使用 <ejb-name> beanHome = (zcxejb1Home)
PortableRemoteObject.narrow(objref,ejbtestHome.class);
……
```

【演示 2】：使用 jboss.xml 文件。

在缺省情况下 JBoss 的 JNDI 服务将通过 ejb-jar.xml 中<ejb-name>XXX</ejb-name>中的 XXX 来使用 EJB 的 home interface。但是如果有多个 EJB 对象在相同的 ejb.jar 包中，在 ejb-jar.xml 中通过<ejb-name>XXX</ejb-name>就可能不能很好的表示某一个 EJB 对象，所以一般希望可以提供一些附加的信息，例如采用这样的格式“[应用名]/[bean 名]”来引用一个 EJB 对象。这时 JNDI 服务就可能不能正确地找到 EJB 对象，就需要使用 jboss.xml 文件，通过它实现 jndi 名到 EJB 名的重定向。这个文件必须和 ejb-jar.xml 一起放到 META-INFO 目录下。

```
ejb-jar.xml:
<ejb-jar>
……
<ejb-name>ejbtest</ejb-name>
……
</ejb-jar>
jboss.xml:
<jboss>
<ejb-name>ejbtest</ejb-name>
<jndi-name>example/ejbtest</jndi-name>
</jboss>
```

客户端：需要使用新的 jndi 名进行 ejb 定位。

```
……
InitialContext ctx = new InitialContext();
Object objref = ctx.lookup("example/ejbtest"); // 使用 <jndi-name> beanHome =
(zcxejb1Home) PortableRemoteObject.narrow(objref,ejbtestHome.class);
……
```

第二种方式：

把 EJB 和 Web 应用包装成一个企业应用包。部署 ear 时可以简单地把通过 application.xml 描述 ejb.jar 包和 Web 应用的 war 包，同样在 web.war 中不需要提供特殊的信息就可以和第一种方式一样使用 EJB 对象。客户端代码也不需要进行改动。这里不再举例说明。

此外可以进行更进一步的部署，这时需要使用到 web.xml 和 jboss-web.xml。jboss-web.xml 是 JBoss 提供的一个针对 Web 应用进行配置的文件。jboss-web.xml 和 web.xml 一起放在 Web 应用的 WEB-INF 目录下。

【演示 1】：改动 web.xml 文件，添加<ejb-ref>标记，不使用 jboss-web.xml 文件。

（注意<ejb-ref>包括内部引用和外部引用。如果是同一个单元的可以通过<ejb-link>直接进行引用，而不用提供其他信息）

```
web.xml:
……
<ejb-ref>
    <ejb-ref-name>ejb/ejbtest</ejb-ref-name>  <! -采用 sun 推荐的命名方式-->
        <ejb-ref-type>Session</ejb-ref-type>
        <home>org.zcx.test.zcxejb1Home</home>
        <remote>org.zcx.test.zcxejb1</remote>
        <ejb-link>ejbtest</ejb-link>    <!--必须和被应用的 ejb-name 匹配-->
</ejb-ref>
……
```

客户端：由于在 web.xml 中引入 ejb 引用描述这时 ejb 定位发生了变化：

```
……
InitialContext ctx = new InitialContext();
Object objref = ctx.lookup("java:comp/env/ ejb/ejbtest "); //使用 java:comp/env 命名空间
beanHome=(zcxejb1Home)PortableRemoteObject.narrow(objref,ejbtestHome.class);
……
```

【演示 2】：联合使用 web.xml 和 jboss-web.xml。

```
web.xml
……
<ejb-ref>
<ejb-ref-name>ejb/ejbtest</ejb-ref-name>  <!--采用 sun 推荐的命名方式-->
      <ejb-ref-type>Session</ejb-ref-type>
      <home>org.zcx.test.zcxejb1Home</home>
      <remote>org.zcx.test.zcxejb1</remote>
</ejb-ref>
……
jboss-web.xml
<ejb-ref>
<ejb-ref-name> ejb/ejbtest </ejb-ref-name>
<jndi-name> example/ejbtest </jndi-name> <!-- 这里对应 ejb 对象的 jndi 名-->
</ejb-ref>
```

客户端：

```
nitialContext ctx = new InitialContext();
Object objref = ctx.lookup("java:comp/env/ ejb/ejbtest "); beanHome = (zcxejb1Home)
PortableRemoteObject.narrow(objref,ejbtestHome.class);
```

以上的所有演示说明了 ejb-jar.xml、jboss.xml、web.xml、jboss-web.xml 之间的基本关系和它们的使用方式。它们是部署各类 EJB 对象的关键文件。

实例 149　Eclipse 开发 J2EE 配置

本实例演示如何在 Eclipse 中开发 J2EE 所需的工具及整合。

1. 下载所需要的软件

j2eesdk1.4SDK：http://java.sun.com/j2ee/1.4/download.html，下载 j2eesdk-1_4-windows.exe。

Eclipse3.0：http://www.eclipse.org/，下载 eclipse-SDK-3.0-win32.zip。

lomboz3.0：http://www.objectlearn.com/index.jsp 上下载 lomboz.3.zip 和 emf-sdo-runtime 2.0.0.zip。

jboss3.2.3：http://www.jboss.org/index.html，下载 jboss-3.2.3.zip。

2. 配置

安装 j2eesdk1.4SDK，假设安装在 C:\Sun\AppServer 目录下，

（1）添加环境变量 ClassPath；C:\Sun\AppServer\jdk\lib\tools.jar；C:\Sun\AppServer\jdk\lib\ dt.jar；C:\Sun\AppServer\lib\j2ee.jar。再在环境变量 Path 添加 C:\Sun\AppServer\jdk\bin；C:\Sun\AppServer\bin。

（2）安装 Eclipse3.0，将 eclipse-SDK-3.0-win32.zip 解压缩，假设解到 D:\Eclipse3 目录下，运行 eclipse.exe。

（3）安装 lomboz，把文件 lomboz.3.zip 和 emf-sdo-runtime-2.0.0.zip 解压缩后，按照解压后的目录对应拷贝到 D:\Eclipse3\plugins 和 D:\Eclipse3\ D:\Eclipse3\features 目录下，再次运行 eclipse.exe，在菜单中选择 File-new-other，出现如图 15-6 所示的界面。

（4）安装 jboss，将 jboss-3.2.3.zip 解压缩到 D:\jboss-3.2.3 目录下。

（5）配置 Eclipse3.0，在菜单中选择 windows-preferences，分别按照如图 15-7、图 15-8、图 15-9 所示的界面配置 Eclipse。

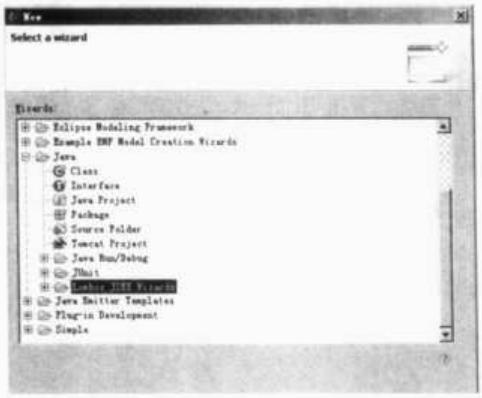


图 15-6

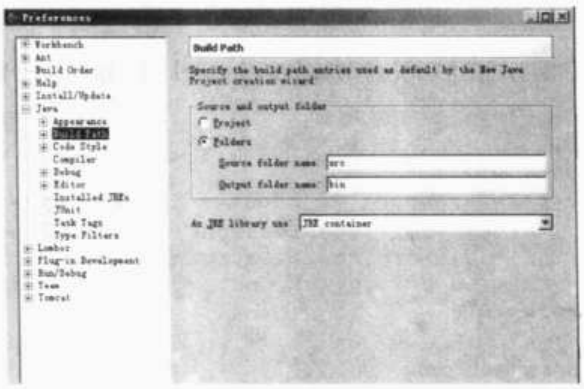


图 15-7

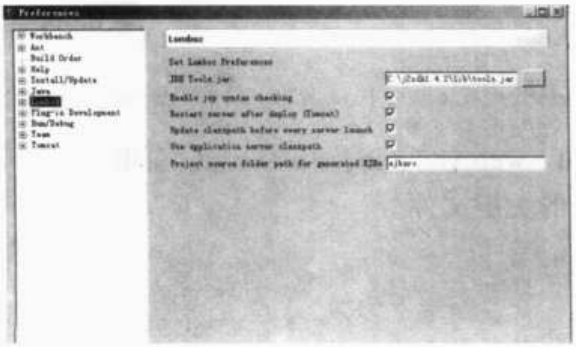


图 15-8

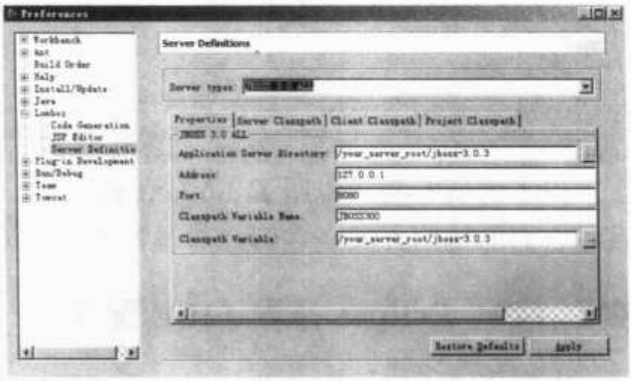


图 15-9

实例 150　安装、配置和执行 MIDlet

本实例演示如何使用安装和配置和执行 MIDlet。

1. 下载

从 http://java.sun.com/products/j2mewtoolkit/download-2_1.html 上下载 j2me_wireless_toolkit-2_1_01-windows.exe。

另外还需先下载安装 jdk1.4 或更高版本。

下载并安装该软件，会在“开始”菜单的“所有程序”中添加了 j2me_wireless_toolkit-2_1_01 项。

2. 部署

如果要创建一个 HelloMIDlet 文件，首先将 WTK 的 lib 目录下的 jar 包设置到变量 CLASSPATH，或者添加到 IDE 的类路径中，才能正确编写要部署的.java 文件。然后选择“开始”→“程序”→“j2me_wireless _toolkit-2_1_01”→“Ktoolbar”进行部署。

在 ktoolbar 界面中，单击“New Project”按钮，在弹出的窗口中输入如图 15-10 所示的数据。

然后单击“Create Project”，将出现图 15-11 所示的工程设置窗口。在该窗口中可以设置工程的各种属性，此时先按默认设置，单击“OK”。

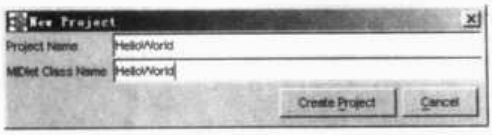


图 15-10

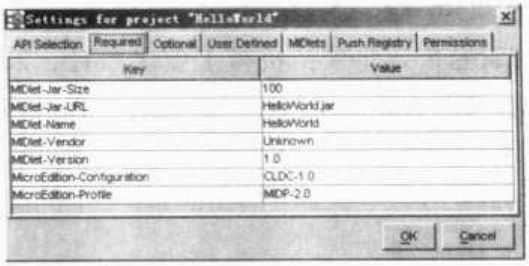


图 15-11

WTK 把所有工程都放在 WTK21\apps\目录下。创建工程后检查该目录，会发现多出一个 HelloWorld 文件夹。将 java 文件复制到 HelloWorld\src 目录下，然后单击 ktoolbar 上的“Build”按钮编译源程序。最后单击“Run”就可以看到运行结果。

小结

本章介绍了 Java 各种服务器的安装和配置，这些服务器都是现在比较流行的、可靠的服务器，读者一般可以从服务器的帮助文档中获得关于该服务器的安装和配置信息。